UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Gabriel Pordeus Santos

IMPLEMENTAÇÃO DE CONTROLE DE ACESSO FEDERADO BASEADO EM ATRIBUTOS PARA AMBIENTES DE NUVEM APACHE CLOUDSTACK

Florianópolis, SC
2024

Gabriel Pordeus Santos

IMPLEMENTAÇÃO DE CONTROLE DE ACESSO FEDERADO BASEADO EM ATRIBUTOS PARA AMBIENTES DE NUVEM APACHE CLOUDSTACK

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciências da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientadora: Profa. Dra. Carla Merkle Westphall
Coorientador: Me. Rafael Weingärtner

Florianópolis, SC
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Santos, Gabriel Pordeus
Implementação de controle de acesso federado baseado em atributos para ambientes de nuvem Apache CloudStack / Gabriel Pordeus Santos ; orientadora, Carla Merkle Westphall, coorientador, Rafael Weingärtner, 2024.
161 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

1. Ciências da Computação. 2. Controle de acesso. 3. Identidade federada. 4. Computação em nuvem. I. Westphall, Carla Merkle. II. Weingärtner, Rafael. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Gabriel Pordeus Santos

IMPLEMENTAÇÃO DE CONTROLE DE ACESSO FEDERADO BASEADO EM ATRIBUTOS PARA AMBIENTES DE NUVEM APACHE CLOUDSTACK

Este(a) Trabalho de Conclusão de Curso foi julgado adequado(a) para obtenção do Título de Bacharel em Ciências da Computação, e foi aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação do Departamento de Informática e Estatística, Centro Tecnológico da Universidade Federal de Santa Catarina.

Florianópolis, SC, 02 de dezembro de 2024.

Nome do Coordenador(a), Dra.
Coordenador(a) do Programa de
Graduação em Ciências da Computação

Banca Examinadora:

Profa. Dra. Carla Merkle Westphall
Orientadora

Me. Rafael Weingärtner
Coorientador

Dr. Guilherme Arthur Gerônimo
Avaliador

AGRADECIMENTOS

À minha família (em especial minha irmã Mariana, que foi de imensa ajuda na elaboração do texto), cujo apoio se provou a razão de concluir este trabalho e, portanto, o curso.

Agradeço também aos meus orientadores, Rafael Weingärtner e professora Carla Merkle Westphall, pelo amparo no planejamento e organização do trabalho, o que proporcionou direção e possibilitou que eu enxergasse a "luz no fim do túnel".

“E o fim é belo, incerto
Depende de como você vê
O novo, o credo
A fé que você deposita em você e só”
(Fernando Anitelli)

RESUMO

Este trabalho tem como objetivo desenvolver uma solução para o Apache CloudStack que permita a recepção de atributos dos provedores de identidade durante o processo de Single Sign-On via SAML, incorporando-os ao controle de acesso. Conceitos relacionados à gestão de identidade e computação em nuvem são levantados e contextualizados. Então, o fluxo de autenticação SAML presente no Apache CloudStack é diagramado e uma especificação é criada para estender o processo de controle de acesso, permitindo a criação automática de contas e usuários inexistentes. Funcionalidades existentes são reaproveitadas para possibilitar o mapeamento dos atributos recebidos do IdP para uma estrutura pré-definida. São realizadas adaptações para manter a compatibilidade com o comportamento prévio do sistema. Por fim, a solução é implementada e testada. A solução aqui desenvolvida abre caminho para futuras inovações e melhorias na integração entre o Apache CloudStack e provedores de identidade, além de no seu próprio processo de autorização interno.

Palavras-chaves: ABAC. Identidade federada. SAML. Apache CloudStack.

ABSTRACT

This bachelor's thesis aims to develop a solution for Apache CloudStack that allows receiving attributes from identity providers during the SAML Single Sign-On process, incorporating them to access control. Concepts related to identity management and cloud computing are raised and contextualized. Then, current SAML authentication flow is diagrammed. A specification is created to extend the access control process, allowing for automatic creation of non-existing users and accounts. Existing functionalities are reused to allow for the attributes received from the IdP to be mapped to a pre-defined structure. Adaptations are made to maintain compatibility to the system previous behavior. Finally, the solution is implemented and tested. The developed solution makes room for future innovations and improvements in the integration of Apache CloudStack and identity providers, on top of its own internal authorization process.

Keywords: ABAC. Federated identity. SAML. Apache CloudStack.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ABAC	*Attribute-Based Access Control*
ACS	Apache CloudStack
API	*Application Programming Interface*
DAO	*Data Access Object*
FIM	*Federated Identity Management*
IaaS	*Infrastructure as a Service*
IAM	*Identity and Access Management*
IdM	*Identity Management*
IdP	*Identity Provider*
JSON	*JavaScript Object Notation*
MS	*Management Server*
OIDC	*OpenID Connect*
RBAC	*Role-Based Access Control*
RBAC-A	RBAC com atributos
RP	*Relying Party*
SAML	*Security Assertion Markup Language*
SLO	*Single Logout*
SP	*Service Provider*
SSO	*Single Sign-On*
UA	*User Agent*

SUMÁRIO

1 INTRODUÇÃO

A gestão de identidade acontece através da coordenação de diversos elementos, exigindo planejamento e desenvolvimento para atender às demandas das regras de negócio, da segurança e da experiência de usuário (Wilson; Hingnikar, 2019). Ela é parte essencial na aceitação dos serviços em nuvem, exigindo controles de acesso que sejam flexíveis e granulares (Indu; Anand; Bhaskar, 2018).

Nos ambientes de rede modernos, a aplicação tradicional de usuário e senha como credenciais para sistemas individuais é limitada, ineficiente e potencialmente insegura. Nestes contextos, portanto, a gestão de identidade é um componente crítico para prover segurança, flexibilização e uniformidade nos acessos aos serviços, através de *Single Sign-On*, *Single Logout* e implementando-se federações (ITU-T, 2009b).

O Apache CloudStack é uma plataforma de orquestração de nuvem de código aberto, parte da fundação Apache e aberta a contribuições da comunidade (ASF, 2024a). O projeto suporta *Single Sign-On* por SAML, mas o seu escopo de aplicação é limitado, pois ignora a possibilidade de receber informações úteis do provedor de identidade: durante o *login*, o ACS utiliza apenas um único atributo dentre todos os recebidos como chave para identificar o usuário logado.

Este trabalho contextualiza, propõe e implementa uma extensão ao procedimento, possibilitando que o ACS identifique os atributos presentes na resposta SAML e os considere no *login* e na sua gestão de identidades, através da criação e atualização de contas de usuário.

No âmbito de controle de acesso, a utilização de atributos permite atingir flexibilidade e escalabilidade que são necessárias em um ambiente de computação em nuvem (Cha; Seo; Kim, 2012). Quando combinado com o controle de acesso baseado em papéis, se torna também auditável e compreensível (Hu et al., 2015).

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver uma solução que habilite o Apache CloudStack a receber atributos dos provedores de identidade durante o processo de *Single Sign-On* (por SAML) e considere-os no seu processo de controle de acesso.

1.1.1 Objetivos Específicos

Os objetivos específicos que podem ser listados são:

1. Levantar e situar os conceitos de gestão de identidade e computação em nuvem relacionados à proposta;
2. Diagramar o fluxo de autenticação com SAML presente no Apache CloudStack;

3. Criar uma especificação para estender este fluxo de controle de acesso, permitindo a criação de contas e usuários que não existam; e,

4. Implementar e testar a solução elaborada.

1.2 ESCOPO

A solução abordará apenas atributos recebidos quando realizado *login* por SAML 2.0. Apesar do Apache CloudStack permitir o acesso por OAuth2, ele ainda não será estendido. Por enquanto, não serão implementados atributos internos, e sim considerados apenas os provenientes do IdP externo. Também não será considerada a plataforma no papel de provedor de identidade, quando é realizado o *login* padrão por usuário e senha.

1.3 ORGANIZAÇÃO DO TEXTO

Os conceitos que fundamentam a proposta de implementação são apresentados no capítulo 2. No capítulo 3, são mostrados alguns trabalhos relacionados e referenciados. No capítulo 4, o planejamento da solução é especificado. O capítulo 5 descreve o processo de implementação. Por fim, o capítulo 6 reúne as conclusões e possibilidades para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, apresentaremos os conceitos que estabelecem a base para a proposta de implementação: no que consiste gestão de identidade, suas ferramentas e protocolos utilizados; e do que se trata orquestração em nuvem, bem como duas populares plataformas de código aberto.

2.1 GESTÃO DE IDENTIDADE

Uma identidade é um conjunto de atributos associados a uma entidade (seja um usuário, elemento de rede, aplicação de software, entre outros) em um contexto específico (Wilson; Hingnikar, 2019; ITU-T, 2009b). No escopo deste trabalho, consideramos este contexto como um sistema de informação.

A gestão de identidade (*Identity Management* – IdM) refere-se às funcionalidades e características (como a administração, manutenção, comunicação, aplicação de políticas, etc.) que confirma identidades, assegura suas informações e viabiliza suas aplicações nos contextos de segurança e de negócios (ITU-T, 2009b).

Wilson e Hingnikar (2019) caracterizam um sistema de IdM como parte importante de um modelo de segurança abrangente: a coleção dos serviços que criam, modificam e removem identidades, bem como autenticam e autorizam o acesso de recursos. Bertino e Takahashi (2011) classificam as partes envolvidas na IdM em usuários (entidades com identidades registradas), provedores de identidade e provedores de serviço (Bhargav-Spantzel et al., 2007).

Um provedor de identidade (*Identity Provider* – IdP) é a entidade que cria, mantém e gerencia informações confiáveis de identidade, oferecendo serviços relacionados a essa confiabilidade (ITU-T, 2009b). Chadwick (2009) descreve-o como uma autoridade que confere atributos a usuários e provê asserções (alegações de que um usuário específico possui determinado atributo), além de realizar autenticações.

Um provedor de serviço (*Service Provider* – SP) é a entidade que precisa conhecer o quão confiáveis são as asserções vinculadas às credenciais fornecidas por seus usuários, para que possa responder às suas requisições (Bertino; Takahashi, 2011). Credenciais são as informações utilizadas para autenticar e autorizar um usuário (ITU-T, 2009b).

2.1.1 Autenticação e autorização

Chamamos de autenticação a instauração de confiança na identidade de um usuário (ITU-T, 2009b). No relacionamento com um SP, ela produz as garantias necessárias para a reivindicação de uma identidade por um usuário (ITU-T, 1996).

Na sua forma mais simples, ambos usuário e SP lidam diretamente com as

informações necessárias para o estabelecimento da confiança, requerendo que as credenciais sejam acordadas diretamente entre eles. Dessa forma, as credenciais são guardadas em cada SP, para cada usuário, resultando em um crescimento quadrático no armazenamento destas credenciais. Alternativamente, as credenciais podem ser obtidas de um terceiro previamente confiável. É possível que este terceiro esteja envolvido em todo o processo, sem necessariamente mediar diretamente a comunicação entre as partes (ITU-T, 1996).

A autenticação sozinha não abrange a delimitação dos privilégios de um usuário. Para isso, é preciso o processo de autorização (ITU-T, 2009a). A recomendação X.800 (CCITT, 1991, p. 4, tradução nossa) define autorização como "a concessão de direitos, incluída a concessão de acesso baseada em direitos de acesso". Esta definição é criticada por Jøsang (2017, p.7, tradução nossa), que propõe a alternativa "autorização é a especificação de políticas de acesso" e a complementa descrevendo controle de acesso, apresentado na seção seguinte, como a execução destas políticas.

2.1.2 Controle de acesso

O processo de controle de acesso provê a base para políticas de segurança em grande parte da área de segurança de computadores (Lampson et al., 1992). Ele decide se pode-se confiar na entidade que realiza uma requisição: por exemplo, se um usuário pode solicitar uma operação de leitura (Abadi, 2009).

A gestão de identidade, quando abrange o controle de acesso, pode também ser chamada de gestão de identidade e acesso (*Identity and Access Management* – IAM) (Weingärtner, 2014).

Dentre as diversas formas de modelar um sistema de controle de acesso, encontramos o baseado em papéis (*Role-Based Access Control* – RBAC) e o baseado em atributos (*Attribute-Based Access Control* – ABAC) (Indu; Anand; Bhaskar, 2018).

No modelo de controle de acesso RBAC, são montados conjuntos de permissões, chamados de *roles* (papéis ou funções), de acordo com as regras de negócio e de segurança da organização. Cada usuário, por sua vez, tem atribuído a si uma *role*, que determinará o que consegue acessar (Coyne; Weil, 2013).

Ferraiolo et al. (2001) descrevem quatro componentes do modelo RBAC: RBAC principal; RBAC hierárquico; relações com separação estática de tarefas; e relações com separação dinâmica de tarefas. O modelo principal reúne as características necessárias para que um modelo seja considerado RBAC: as atribuições de *roles* a usuários e permissões a *roles*. Além disso, o modelo principal abrange a existência de sessões, que determinam quais *roles* estão ativas em determinada interação do usuário com o recurso.

A Figura 1 ilustra o modelo RBAC principal, com os elementos: usuários, *roles*, atribuição de usuários (*User Assignment*), atribuição de permissões (*Permission*

Assignment), objetos (OBS), operações (OPS) e permissões (PRMS).

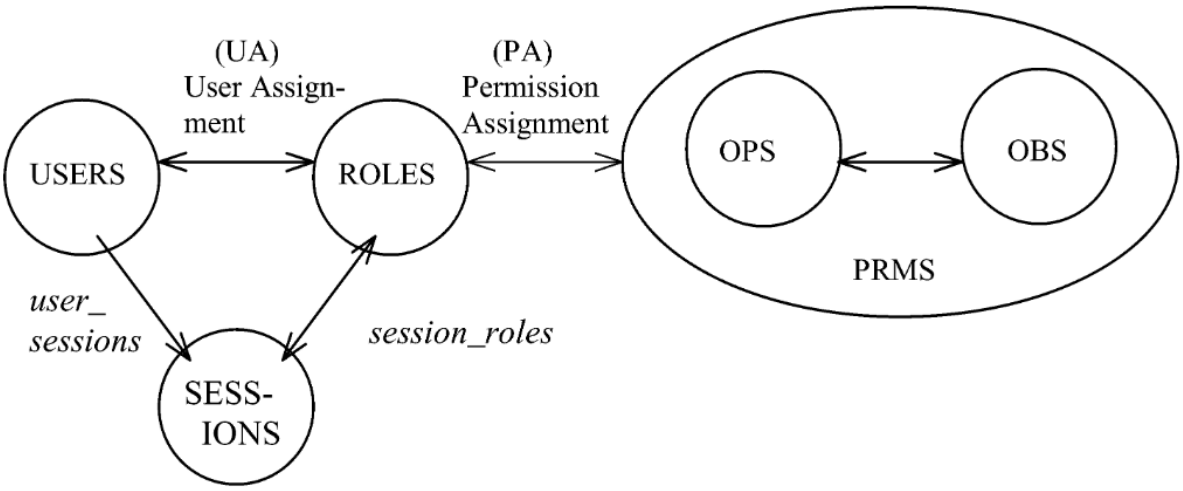

Figura 1 – Modelo principal de RBAC (Ferraiolo et al., 2001, p. 232)

O RBAC é facilmente auditável, bastando verificar as permissões que a *role* possui em certo momento, mas exige que elas sejam bem planejadas e encontra limitações ao lidar com características dinâmicas (Coyne; Weil, 2013).

Em modelos de controle de acesso ABAC, são analisadas diversas informações do usuário e de contexto, como hora e local (Bertino; Takahashi, 2011). Não é necessário planejar *roles*, mas os atributos (potencialmente numerosos) precisam ser selecionados por especialistas. O processo de auditoria é complicado, uma vez que é preciso conhecer o histórico de quais atributos foram atribuídos a quais usuários, o que pode se mostrar inviável (Coyne; Weil, 2013).

2.1.3 Federação

No contexto de IdM, uma federação é o nome dado à associação que abrange diversos SPs e IdPs. Neste paradigma, mantém-se a independência entre os membros, ao mesmo tempo em que facilita-se o compartilhamento de informações de identidades para viabilizar os serviços federados (ITU-T, 2009b).

Para o estabelecimento de uma federação, é preciso que os integrantes tenham certo grau de confiança entre si para que a comunicação funcione. Quando a federação compartilha credenciais de autenticação e autorização, temos a gestão de identidade federada (*Federated Identity Management* — FIM) (Chadwick, 2009). Sistemas de FIM costumam adotar o modelo ABAC (Chadwick; Inman, 2009).

Wilson e Hingnikar (2019) caracterizam *Single Sign-On* (SSO) como o termo utilizado para o acesso a diferentes recursos e serviços, com as mesmas necessidades de autenticação, fornecendo as credenciais uma única vez. Os autores explicam que ele fornece aos usuários as conveniências de menos autenticações, de menos senhas para lembrar e da não exposição das suas credenciais às aplicações finais; além disso,

permite a organizações que centralizem suas políticas de autenticação.

Da mesma forma, *Single Logout* (SLO) é a sincronização do encerramento de sessões. Mensagens de *logout* são enviadas aos SPs quando a sessão no IdP é finalizada (Wilson; Hingnikar, 2019).

2.1.4 Protocolos

Security Assertion Markup Language (SAML) é um *framework* para a transmissão de informações de segurança, na forma de asserções sobre os usuários codificadas em XML (ITU-T, 2007). A especificação da versão SAML 2.0 foi publicada em 2005, fornecendo uma solução para identidade federada e SSO em ambientes *web* (Wilson; Hingnikar, 2019).

A Figura 2 contém o diagrama de sequência retratando o fluxo de mensagens descrito pelo protocolo. O agente do usuário (*User Agent* – UA) acessa o recurso no SP (presente como SAML *requester*, ou solicitante SAML) [1]; o SP responde com um redirecionamento ao IdP (presente como SAML *responder*, ou respondente SAML) [2]; o IdP autentica o UA [3]; o IdP responde com um redirecionamento ao SP [4]; o SP retorna o recurso ao UA [5] (ITU-T, 2007, tradução nossa).

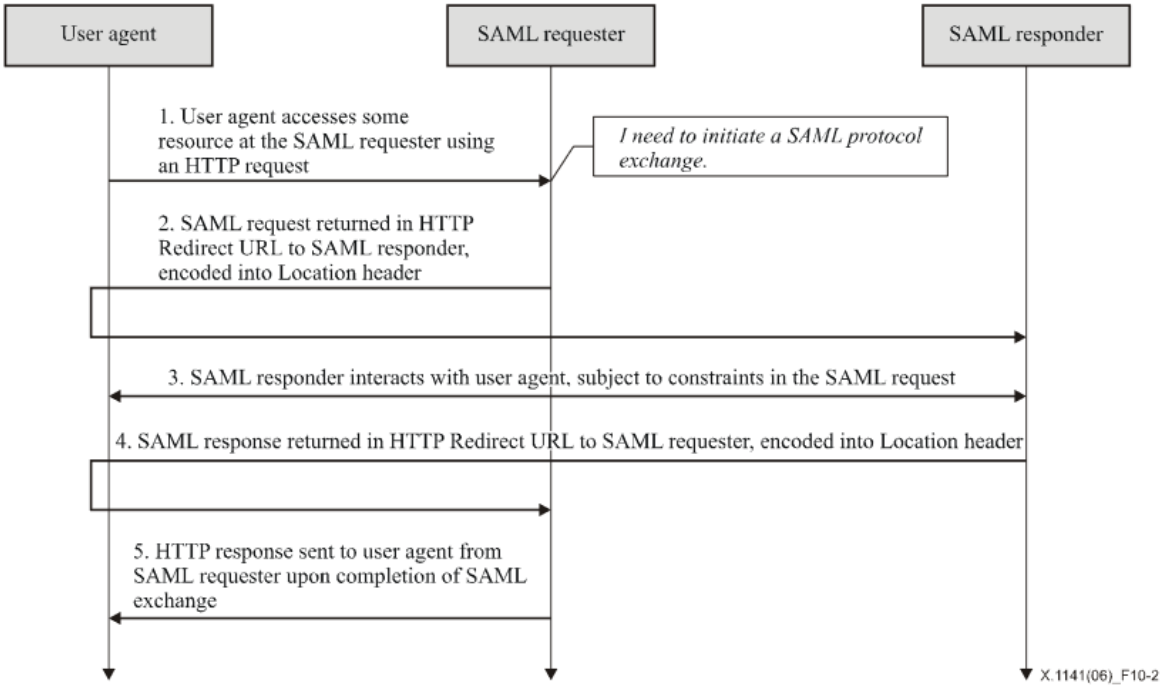

Figura 2 – Diagrama de sequência do fluxo SAML (ITU-T, 2007, p. 104)

Outro protocolo de IAM é o OAuth 2.0. Ele torna possível que um usuário dê permissão a um SP para acessar o conteúdo de um SP terceiro. O usuário realiza a autenticação diretamente com o terceiro e um *token* comprovando que a autenticação foi realizada é repassado ao SP solicitante, que pode então realizar a requisição diretamente ao primeiro SP, em nome do usuário (Wilson; Hingnikar, 2019).

Uma vez que o *token* apenas dá o direito ao acesso a um serviço, mas não contém informações de usuário, este protocolo sozinho não se encaixava bem em serviços gerais de autenticação (Wilson; Hingnikar, 2019).

Wilson e Hingnikar (2019) explicam que, para sanar este problema, o *OpenID Connect* (OIDC) foi elaborado como uma extensão do OAuth 2.0, permitindo repassar informações aos SPs. Dessa forma, se tornou possível não apenas SSO, mas também autorização de API, sendo utilizado por SPs de larga escala, como Google, PayPal e Yahoo, primariamente em serviços voltados ao consumidor.

Mostrado na Figura 3, o fluxo do OIDC começa com o SP (mencionado como *Relying Party* – RP) enviando a requisição ao IdP (chamado de *OpenID Provider* – OP); o IdP autentica o usuário final e obtém autorização; o IdP responde com um *token* ID e um *token* de acesso; com o *token* de acesso, o SP solicita as informações do usuário ao IdP; o IdP retorna as asserções (ou *claims*) (Sakimura et al., 2023).

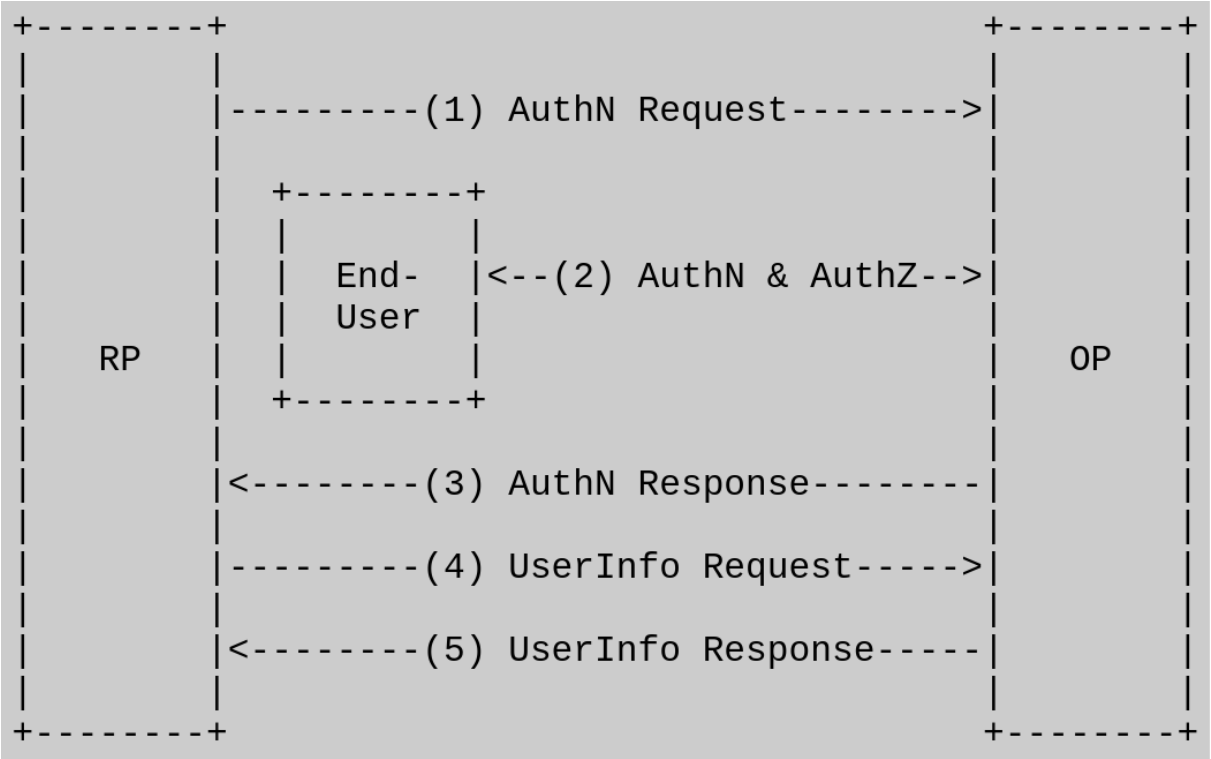

Figura 3 – Visão geral do fluxo OIDC (Sakimura et al., 2023)

2.1.5 Ferramentas

Uma das ferramentas de código aberto mais populares para a implementação de IAM é o Keycloak, preparado para aplicativos de página única, aplicativos móveis e APIs REST, dentre outros. Foi criado em 2014 e, desde então, se tornou um projeto bem estabelecido e presente em amplos cenários, desde pequenos sites a grandes empreendimentos. Trabalha com os protocolos OAuth, OIDC e SAML, sendo facilmente integrado a diversos ambientes. Além disso, é fácil de instalar e é preparado para os

casos de uso mais comuns (como servir de IdP para SSO) sem precisar de adaptações (Thorgersen; Silva, 2021).

Por estas razões e familiaridade com a ferramenta, escolhemos o Keycloak para integrar o ambiente de desenvolvimento. Existem, porém, diversas alternativas de código aberto para implementação de IAM, incluindo Apache Syncope (ASF, 2024b), ZITADEL (ZITADEL, 2024) e Authelia (Authelia, 2024).

2.2 COMPUTAÇÃO EM NUVEM

Computação em nuvem é um modelo de consumo, por rede e sob demanda, de recursos computacionais (como redes, servidores, armazenamento, aplicações e serviços). Eles podem ser instaurados e removidos com o mínimo de esforço e interação (Mell; Grance, 2011).

Mell e Grance (2011, tradução nossa) descrevem a computação em nuvem como reunindo as seguintes características:

> - Autoatendimento sob demanda. Um consumidor pode unilateralmente provisionar recursos computacionais, como tempo de servidor e armazenamento em rede, automaticamente conforme necessário sem necessitar de interação humana com cada provedor de serviço.
> - Amplo acesso de rede. Recursos são disponibilizados via rede e acessados por mecanismos padrões que promovem o uso por plataformas *thin* ou *thick clients* (p. ex. telefones celulares, *tablets*, *laptops* e estações de trabalho).
> - Agrupamento de recursos. Os recursos computacionais do provedor são agrupados para servir múltiplos consumidores utilizando um modelo multilocatário, com diferentes recursos físicos e virtuais dinamicamente alocados e realocados de acordo com a demanda dos consumidores. Existe uma sensação de independência geográfica em que o consumidor geralmente não tem controle ou conhecimento da localidade exata dos recursos providos, mas pode ser capaz de especificar a localidade em um nível mais alto de abstração (p. ex. país, estado ou *datacenter*). Exemplos de recursos incluem armazenamento, processamento, memória e largura de banda de rede.
> - Elasticidade rápida. Recursos podem ser elasticamente provisionados e liberados, em alguns casos automaticamente, para escalar rapidamente externa e internamente de acordo com a demanda. Para o consumidor, os recursos disponíveis para provisionamento com frequência parecem ser ilimitados e podem ser apropriados em qualquer quantidade a qualquer hora.
> - Serviço mensurado. Sistemas de nuvem automaticamente controlam e otimizam o uso de recursos através de uma habilidade de medição com algum nível de abstração apropriada para o tipo de serviço (p. ex. armazenamento, processamento, largura de banda e contas de usuário ativas). O uso de recursos pode ser monitorado, controlado e relatado, provendo transparência para ambos o provedor e o consumidor do serviço utilizado.

Um ambiente de nuvem pode ser privado (com uso exclusivo por uma única organização), comunitário (uso exclusivo por organizações com interesses compartilhados), público (destinado ao público geral) ou híbrido (múltiplas infraestruturas independentes, mas conectadas e que oferecem portabilidade) (Mell; Grance, 2011).

Dentre os tipos de serviços prestados por ambientes de computação em nuvem, está a infraestrutura como serviço (*Infrastructure as a Service* — IaaS), que permite

ao consumidor preparar e executar qualquer *software*, incluindo sistemas operacionais. O consumidor não controla a infraestrutura física diretamente, mas pode configurar certos aspectos de rede (Mell; Grance, 2011).

2.2.1 Ferramentas

A abstração da interação do administrador com os recursos é fornecida por ferramentas de orquestração em nuvem, eliminando a necessidade de que o administrador seja um especialista em todas as áreas da infraestrutura e facilitando a gerência dos recursos aos usuários finais (Martins, 2016).

2.2.1.1 Apache CloudStack

Apache CloudStack (ACS) é um *software* de código aberto, desenvolvido na linguagem Java, que oferece orquestração em nuvem para implementar ambientes de nuvem públicas, privadas e híbridas. Oferece a orquestração de máquinas virtuais, *Network as a Service*, gerência de contas e usuários, contabilidade de recursos e outros processos gerenciais necessários em um ambiente de nuvem. (ASF, 2024a).

Dentre as formas de interação, existe uma interface gráfica *web*, uma ferramenta de linha de comando (CloudMonkey) e uma interface de programação de aplicativo (*Application Programming Interface* — API) nativa. Suporta os virtualizadores VMWare, KVM e XenServer/Xen Cloud Platform. É uma solução que pode ser utilizada por uma equipe pequena e, sendo de código aberto, permite extensões e é alavancada pelas inovações desenvolvidas pela comunidade (ASF, 2024a).

O principal serviço é o *Management Server* (MS), que realiza a orquestração do ambiente em si, com a possibilidade de uso do serviço *Agent*, quando utilizado o virtualizador KVM.

Ilustrados na Figura 4, o ACS estabelece os conceitos de (ASF, 2024c):

- Conta – A representação interna de uma entidade consumidora e/ou administradora;
- Usuário – Uma forma de acesso (credenciais) a uma conta. Uma conta precisa ter pelo menos um usuário, mas pode ter mais;
- Domínio – Um agrupamento de contas para a implementação de regras de negócio e segregação de recursos. Pode conter subdomínios, de maneira recursiva.

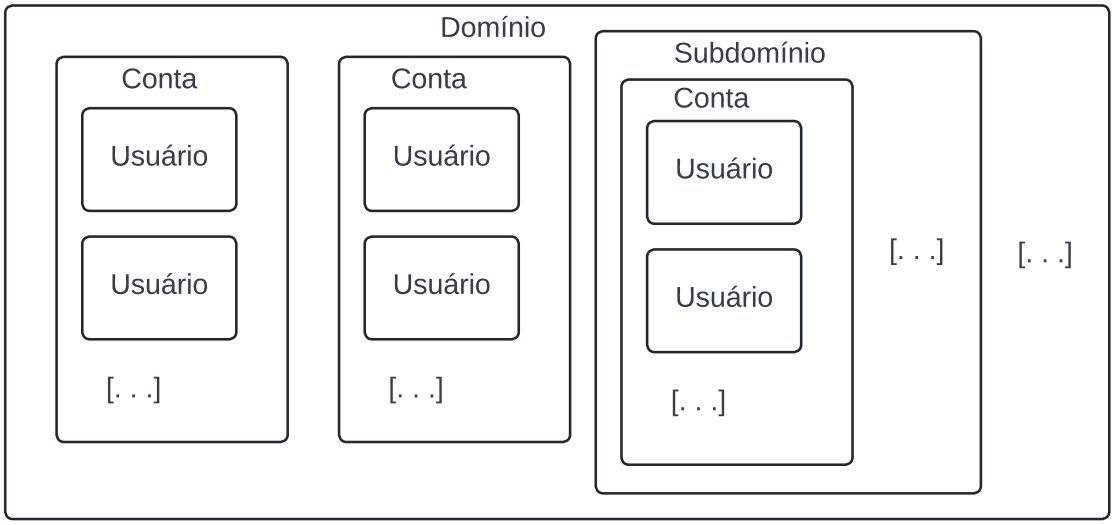

Figura 4 – Diagrama de Usuário, Conta e Domínio no ACS.

O processo de autenticação de um usuário pode se dar por um par de chaves na realização de chamadas de API ou por sessão no navegador, através de credenciais internas (nome de usuário, senha e domínio) ou através de IdPs externos (seja por LDAP, SAML 2.0 ou OAuth2) (ASF, 2024c).

Independentemente da maneira de autenticação, a autorização no ACS usa RBAC e é sempre interna. Cada conta possui um conjunto de permissões denominado *role*. Cada regra de permissão determina o acesso a um *endpoint* da API. Além disso, a conta possui um tipo (usuário, administrador ou administrador de domínio), que pode limitar as ações de um *endpoint* mesmo que a permissão esteja presente (ASF, 2024c). É possível flexibilizar as permissões de uma conta por usuário através das chaves de API, que podem permitir apenas um subconjunto das permissões da *role*.

2.2.1.1.1 SAML 2.0 no ACS

O ACS permite o uso de SAML 2.0 na autenticação de usuários para oferecer SSO e/ou delegar a tarefa de autenticação a uma autoridade central. Entre as configurações necessárias para a integração, estão uma URL que contenha os metadados do IdP, o algoritmo usado para assinatura RSA das requisições SAML (SHA1, SHA256, SHA384 ou SHA512), a URL para onde o IdP deve redirecionar quando autenticado com sucesso e o atributo contendo o nome de usuário a ser logado (ASF, 2024c). A Figura 5 ilustra o fluxo SAML no contexto do ACS.

Para que um usuário consiga realizar o *login* por SAML, é preciso que o administrador autorize-o pela chamada de API `authorizeSamlSso` para um único IdP específico (ASF, 2024c).

O *login* é realizado através de um atributo (por padrão, `uid`, mas pode ser configurado) que deve estar presente na asserção e conter o nome de usuário que está sendo autenticado (ASF, 2024c). No caso de múltiplos usuários com SAML habilitado e com o mesmo nome, pode-se trocar para o usuário desejado em uma lista *dropdown* ou pelo *endpoint* `listAndSwitchSamlAccount`.

2.2.1.1.2 OAuth2 no ACS

Outra opção para SSO no ACS é através de OAuth2. É possível utilizar o Google e/ou o GitHub como IdPs. Para adicionar um IdP (feito pelo *endpoint* `registerOauthProvider`), é preciso fornecer o ID, a URL de redirecionamento e uma chave privada, todos pré-registrados no IdP. O usuário é identificado pelo endereço de *email* e é preciso especificar o domínio na tela de *login* (ASF, 2024c).

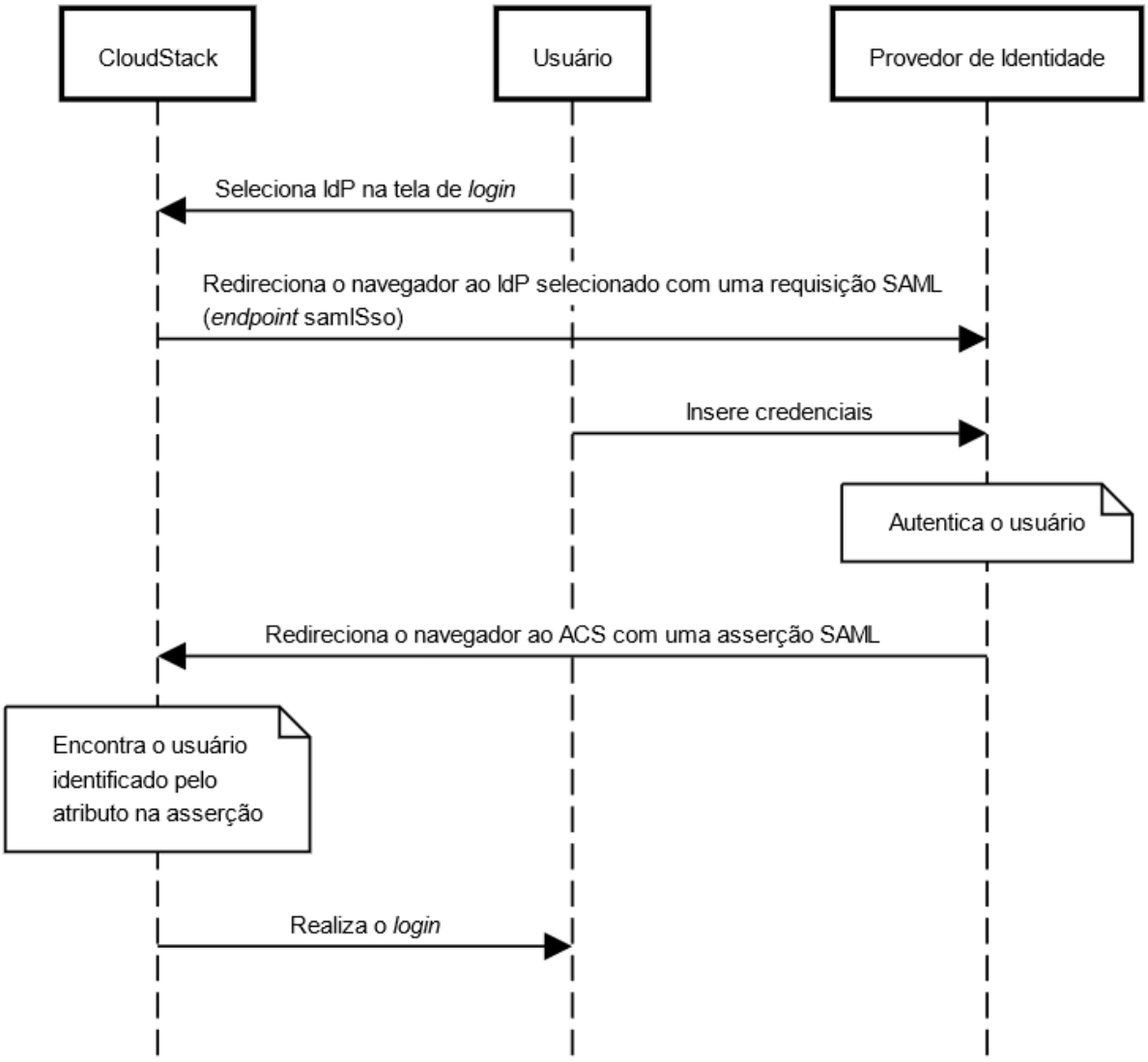

Figura 5 – Fluxo SAML no ACS

2.2.1.2 OpenStack

OpenStack é uma alternativa ao Apache CloudStack, também de código aberto, para orquestração em nuvem. Pode ser operado por um *dashboard*, linhas de comando ou por API. Suporta os virtualizadores KVM, LXC, VMWare, Xen, XenServer (e variantes que usem XAPI), Hyper-V e Virtuozzo (OpenStack, 2024c).

Diferentemente do ACS, que é monolítico, o OpenStack é distribuído em diversos serviços, como o Keystone para IAM, Nova para orquestração de máquinas virtuais, Swift para armazenamento, Neutron para redes definidas por *software*, entre vários outros (OpenStack, 2024c). A Figura 6 ilustra um exemplo de arquitetura OpenStack (que pode variar de acordo com os serviços utilizados em cada ambiente).

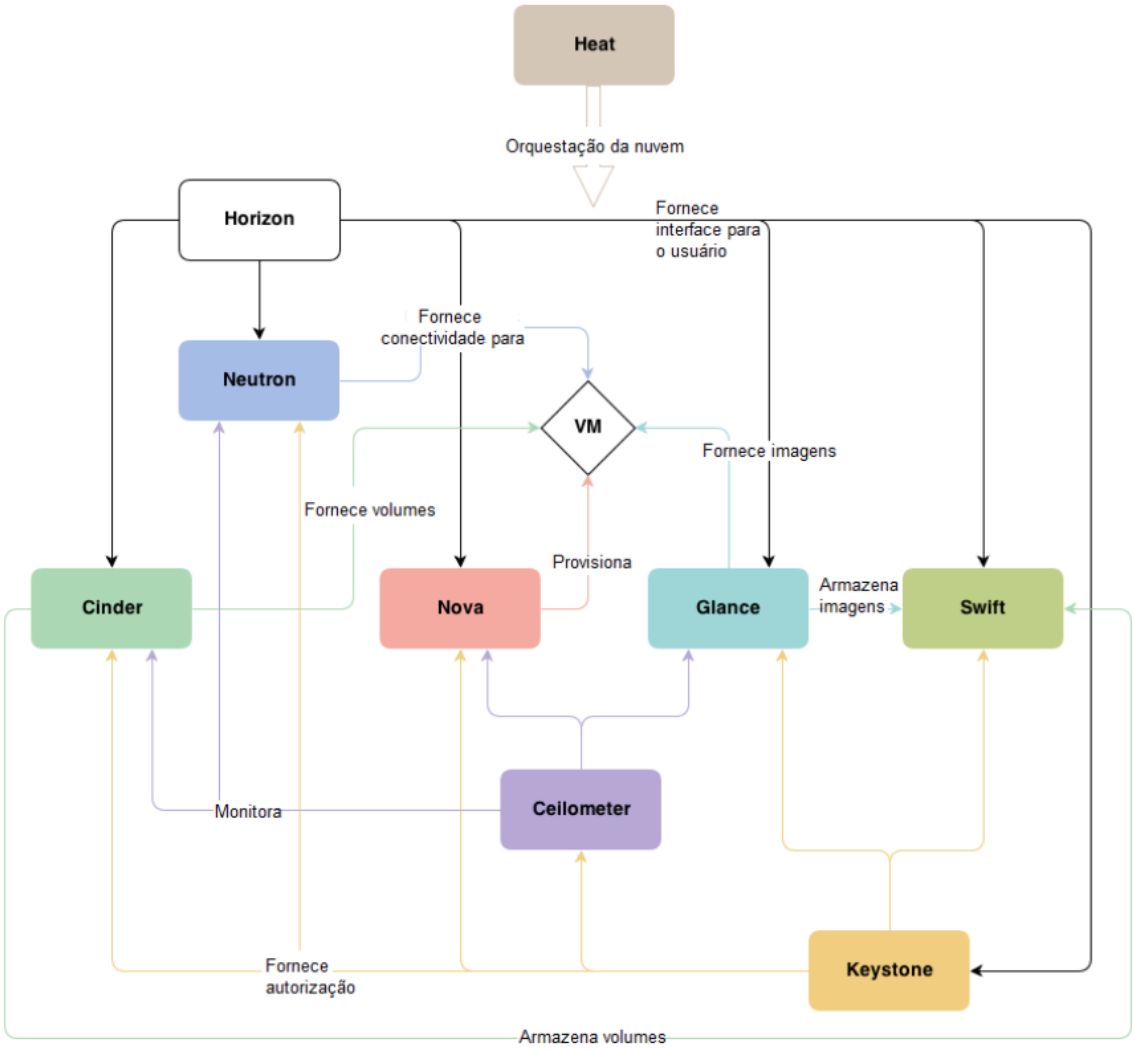

Figura 6 – Exemplo de arquitetura do OpenStack (Miers et al., 2017, p. 227)

Qualquer recurso em um ambiente OpenStack pertence a um projeto. Projetos, por sua vez, são parte de domínios, que determinam o escopo dos seus componentes. Usuários são as unidades consumidoras, que também fazem parte de domínios. *Roles*

determinam as permissões do usuário ao qual são atribuídas, relativas ao escopo de um projeto (OpenStack, 2024b).

Cada serviço OpenStack possui suas próprias políticas de acesso aos seus recursos. As políticas são configuradas pelo administrador, através de um arquivo chamado `policy.yaml` (OpenStack, 2024d).

Atualmente, existe uma proposta de extensão para o OpenStack possibilitando que, de forma dinâmica, defina-se projetos e suas regras recebendo definições do IdP, além de permitir a adição e remoção de *Roles* de acordo com sua presença nos atributos dos usuários disseminados pelos IdPs após o processo de autenticação (Weingärtner, 2020).

2.2.1.2.1 SAML 2.0 no OpenStack

O Keystone pode utilizar SAML 2.0 para prover o controle de acesso com SSO, como ilustrado na Figura 7. O usuário solicita o *login* federado ao Horizon, que redireciona o navegador (com uma URL contendo o protocolo e IdP desejados) ao Keystone (OpenStack, 2024a).

Por sua vez, o Keystone identifica que o usuário não está logado e o navegador é novamente redirecionado, desta vez ao IdP com a requisição SAML. O usuário realiza a autenticação e encaminha a resposta SAML ao SP. A resposta é validada e é feito novo redirecionamento ao Keystone, que devolve um *token* a ser usado pelo usuário para se logar no serviço OpenStack desejado (OpenStack, 2024a).

2.2.1.2.2 OIDC no OpenStack

O OpenStack também suporta OIDC, cuja integração está presente na Figura 8. Quando o usuário solicita o recurso protegido do SP, ele é redirecionado ao IdP para realizar o *login*, tal qual por SAML. O IdP redireciona o usuário de volta ao SP utilizando uma URL (previamente conhecida pelo SP), junto com um código de autorização (OpenStack, 2024a).

Em seguida, o SP realiza uma requisição diretamente ao IdP com o código e recebe um *token* de identificação. Do ponto de vista do Horizon e Keystone, o fluxo é o mesmo, alterando-se apenas os detalhes internos de especificação e formatação (OpenStack, 2024a).

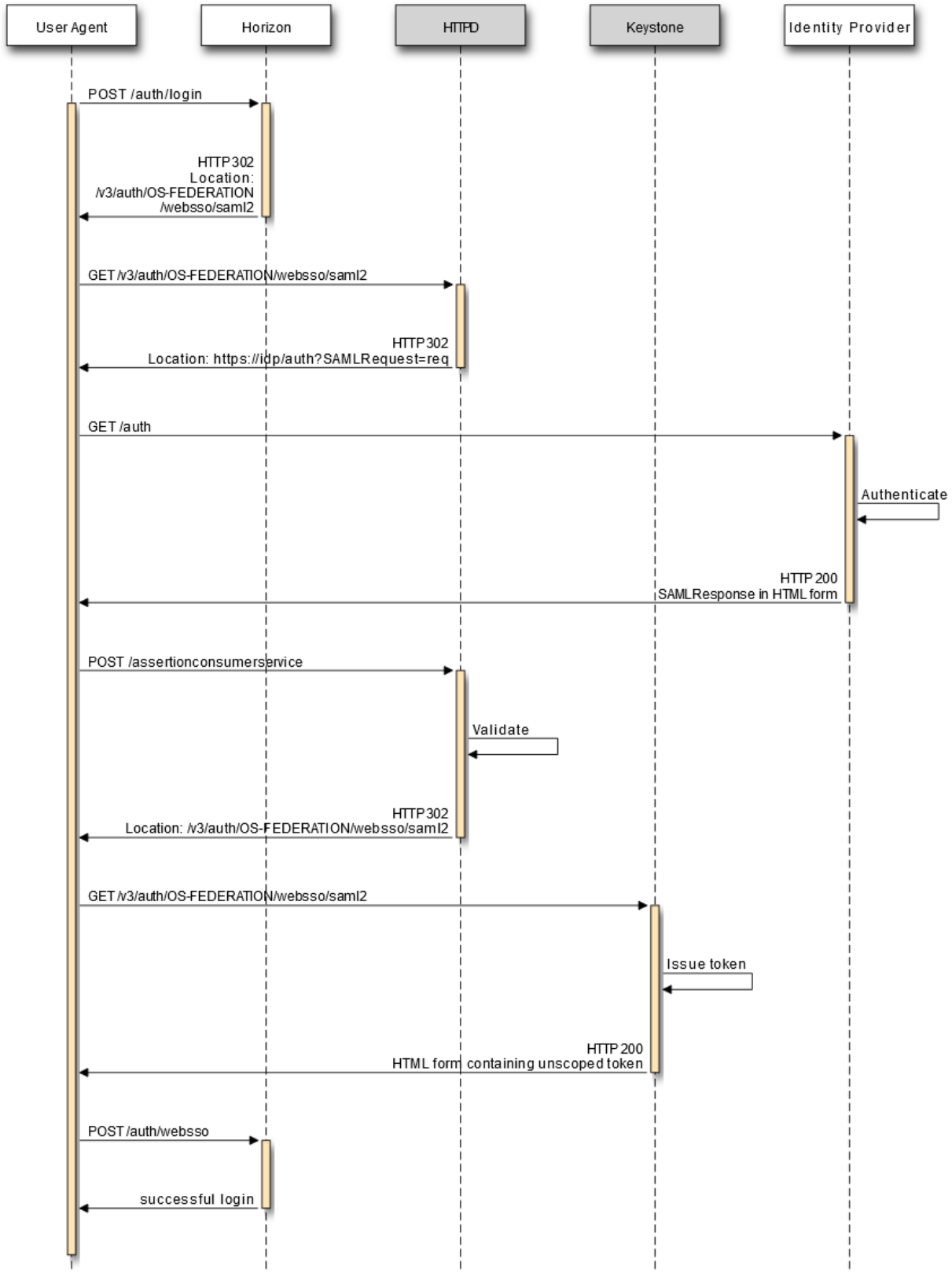

Figura 7 – Fluxo SAML com Keystone e Horizon (OpenStack, 2024a)

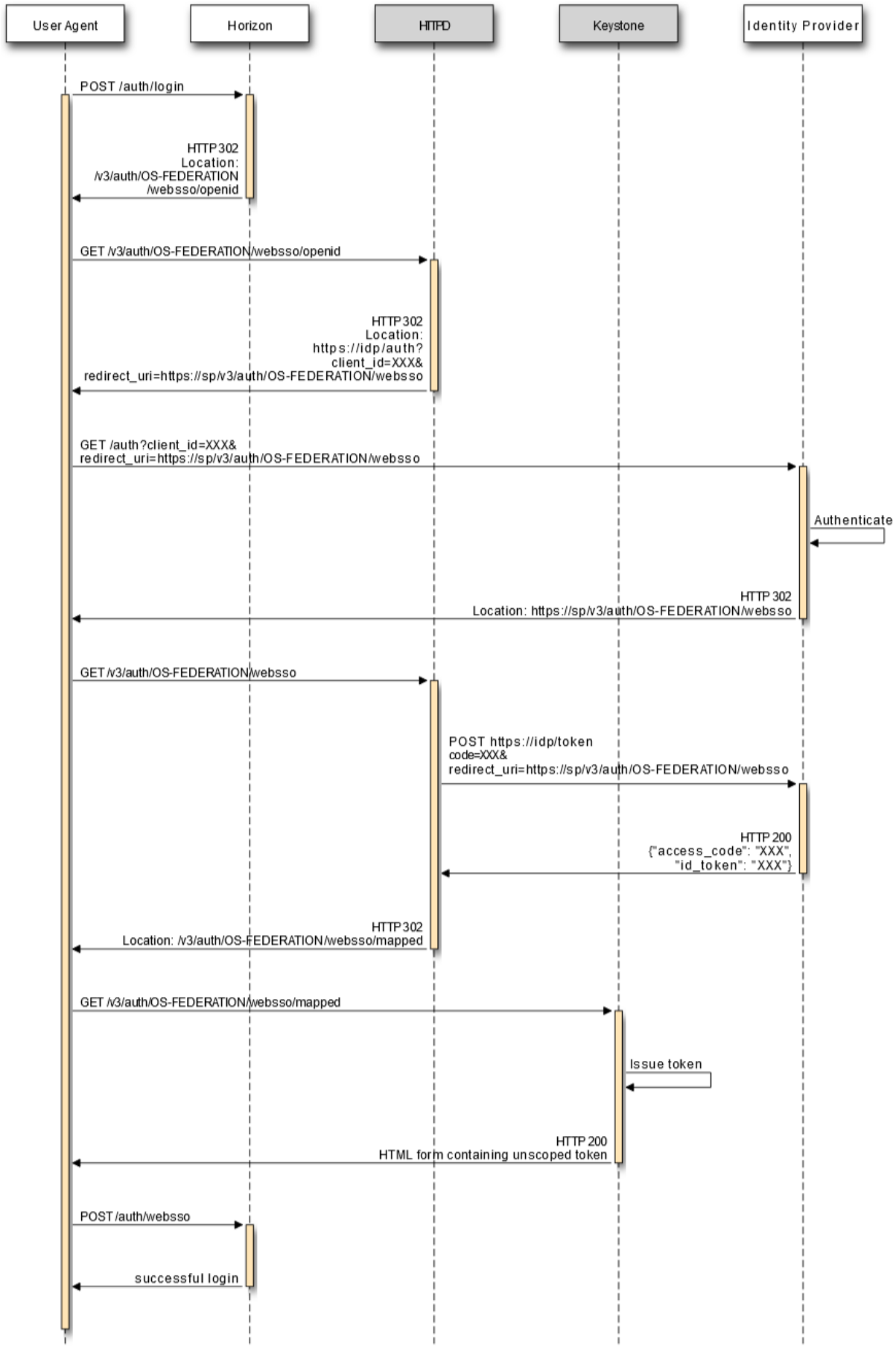

Figura 8 – Fluxo OIDC com Keystone e Horizon (OpenStack, 2024a)

3 TRABALHOS RELACIONADOS

Nesta seção, apresentaremos alguns dos trabalhos semelhantes que existem na literatura e que serviram de referência no desenvolvimento desta proposta.

3.1 ADDING ATTRIBUTES TO ROLE-BASED ACCESS CONTROL (Kuhn; Coyne; Weil, 2010)

O artigo comenta a possibilidade de uma "explosão de *roles*" em soluções RBAC, devido à inflexibilidade na definição nas coleções de permissões, e a contrapõem com as vantagens provindas do ABAC, sugerindo maneiras de combiná-las — reunidas na Tabela 1.

Opção	U	R	A	**Modelo**	**Mapeamento de permissões**
0	0	0	0	indefinido	—
1	0	0	1	ABAC-básico	$A_1, ..., A_n \rightarrow$ perm
2	0	1	0	indefinido	—
3	0	1	1	ABAC-RBAC híbrido	$R, A_1, ..., A_n \rightarrow$ perm
4	1	0	0	ACLs	$U \rightarrow$ perm
5	1	0	1	ABAC-ID	$U, A_1, ..., A_n \rightarrow$ perm
6	1	1	0	RBAC-básico	$U \rightarrow R \rightarrow$ perm
7	1	1	1	RBAC-A, *roles* dinâmicas	$U, A_1, ..., A_n \rightarrow R \rightarrow$ perm
8	1	1	1	RBAC-A, foco em atributos	$U, R, A_1, ..., A_n \rightarrow$ perm
9	1	1	1	RBAC-A, foco em *roles*	$U \rightarrow R \rightarrow A_1, ..., A_n \rightarrow$ perm

*U = ID do usuário; R = *role*; A = atributos

Tabela 1 – Estratégias para combinação de RBAC e ABAC (Kuhn; Coyne; Weil, 2010, p. 80, tradução nossa)

Os autores explicam que as opções 0 e 2 são indefinidas e incluídas para completude e sugerem uma possibilidade de aplicação para as opções 1 e 3 em serviços públicos, com usuários anonimizados.

As abordagens de RBAC com atributos (RBAC-A) são categorizadas em: *roles* dinâmicas, que flexibilizam a atribuição de *roles* incluindo atributos na escolha; focado em atributos, em que a *role* é mais um dos atributos de um usuário; e focado em *roles*, onde os atributos restringem as permissões descritas pela *role*.

3.2 MECANISMOS DE AUTENTICAÇÃO E AUTORIZAÇÃO PARA NUVENS COMPUTACIONAIS: DEFINIÇÃO, CLASSIFICAÇÃO E ANÁLISE DE SOLUÇÕES (Miers et al., 2017)

O artigo do minicurso situa a discussão com a agregação dos conceitos de credenciais, autenticação e autorização, computação em nuvem e SSO para, por fim, realizar um estudo de caso configurando o OIDC no OpenStack.

Os autores apresentam diferentes taxonomias de SSO e as consolidam em uma nova proposta na Figura 9.

Figura 9 – Taxonomia consolidada para SSO (Miers et al., 2017, p. 218)

Para o estudo de caso, o artigo explica o funcionamento dos protocolos OpenID, OAuth e OIDC, além do projeto OpenStack e os seus componentes. Conclui que os usos de SSO facilitam a autenticação e autorização nos serviços *web*, mas que ainda são pouco utilizados em ambientes de nuvem. Os autores ressaltam que, apesar destes mecanismos facilitarem o tratamento de volumes consideráveis de usuários, continua sendo preciso manter a atenção nos aspectos de segurança.

3.3 CONTROLE DE DISSEMINAÇÃO DE DADOS SENSÍVEIS EM AMBIENTES FEDERADOS (Weingärtner, 2014)

A fundamentação teórica reúne temas semelhantes ao deste trabalho: IdM — abordando sua Segurança, modelos de sistemas e ferramentas —, Computação em Nuvem, Privacidade e a legislação envolvida, Controle de Acesso, SSO e Federação. Na revisão bibliográfica, são levantadas trabalhos alternativos que são então comparados de acordo com a presença de certas características (criptografia, reputação, políticas e notificação de uso/disseminação, suporte no processo de disseminação).

A dissertação realiza um levantamento e discussão de problemas de privacidade relacionados a informações de identificação pessoal armazenados em IdPs de federações, além de propor um modelo para mitigar os riscos envolvidos e definir um método para facilitar a integração de IdPs e SPs. A Figura 10 foi utilizada para construir as ameaças que são tratadas pela proposta.

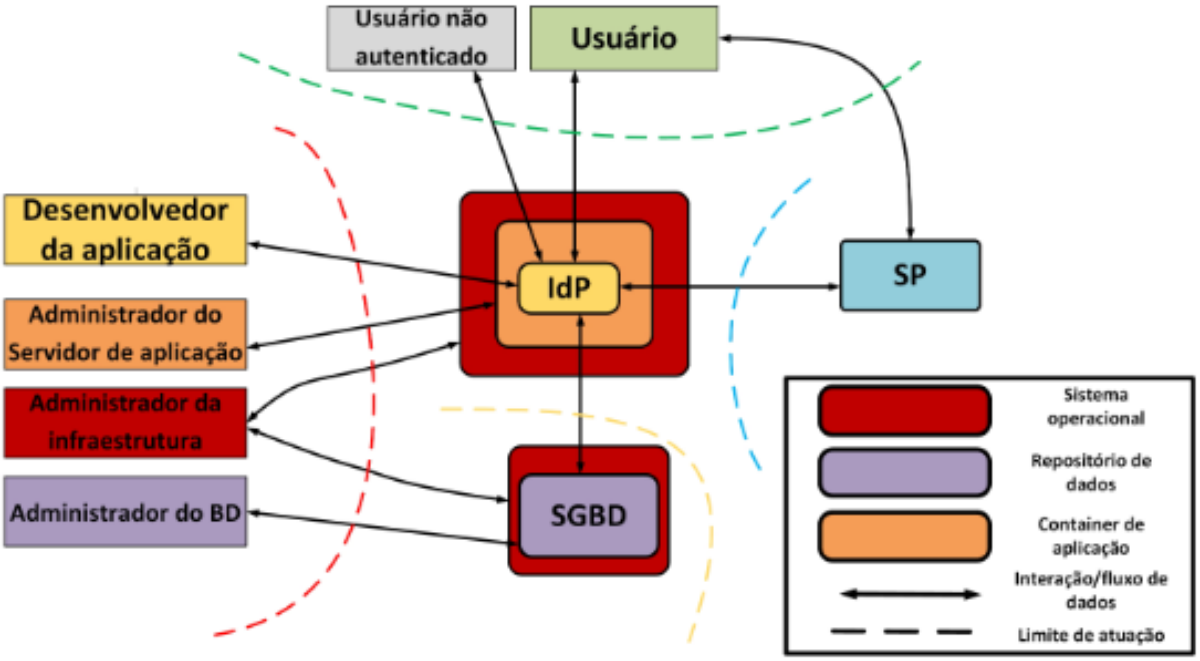

Figura 10 – Diagrama de fluxo de dados das interações com um IdP (Weingärtner, 2014, p. 60)

O autor considera que os ganhos em controle e privacidade na aplicação de criptografia superam as desvantagens na usabilidade. Completa que o tratamento de privacidade deve ter alta prioridade, dado o crescimento de ambientes federados e da computação em nuvem, apesar de, na época, ser pouco explorado justamente nestes ambientes.

3.4 UMA ABORDAGEM DE PRIVACIDADE NO GERENCIAMENTO DE IDENTIDADE NA NUVEM (Werner, 2017)

O autor situa o trabalho com fundamentação teórica e revisão bibliográfica, descrevendo os conceitos relacionados de Privacidade, Computação em Nuvem, Segurança, Identidade, Controle de Acesso, Federação e Tecnologias de IdM, bem como apresentando e comparando diversos trabalhos da área após mapeamento sistemático.

A tese propõe um modelo para melhorar a privacidade dos dados de usuário tratados por IdM em ambiente de nuvem, dando a oportunidade para o usuário de escolher seu próprio perfil de compartilhamento de atributos (entre perfis mais e menos permissivos) e categorizar seus dados entre mais e menos sensitivos.

Também desenvolve um protótipo do processo de negociação dinâmica dos requisitos de privacidade na integração da arquitetura do ambiente de nuvem, deixando claro que a avaliação do desempenho da solução não fazia parte do escopo. Estudos de caso nas áreas médica e de comércio eletrônico são apresentados, tendo resultados positivos com a aplicação do protótipo.

O modelo de IdM com privacidade proposto na tese é retratado na Figura 11, onde as preferências de privacidade são responsabilidade tanto do IdP como do SP.

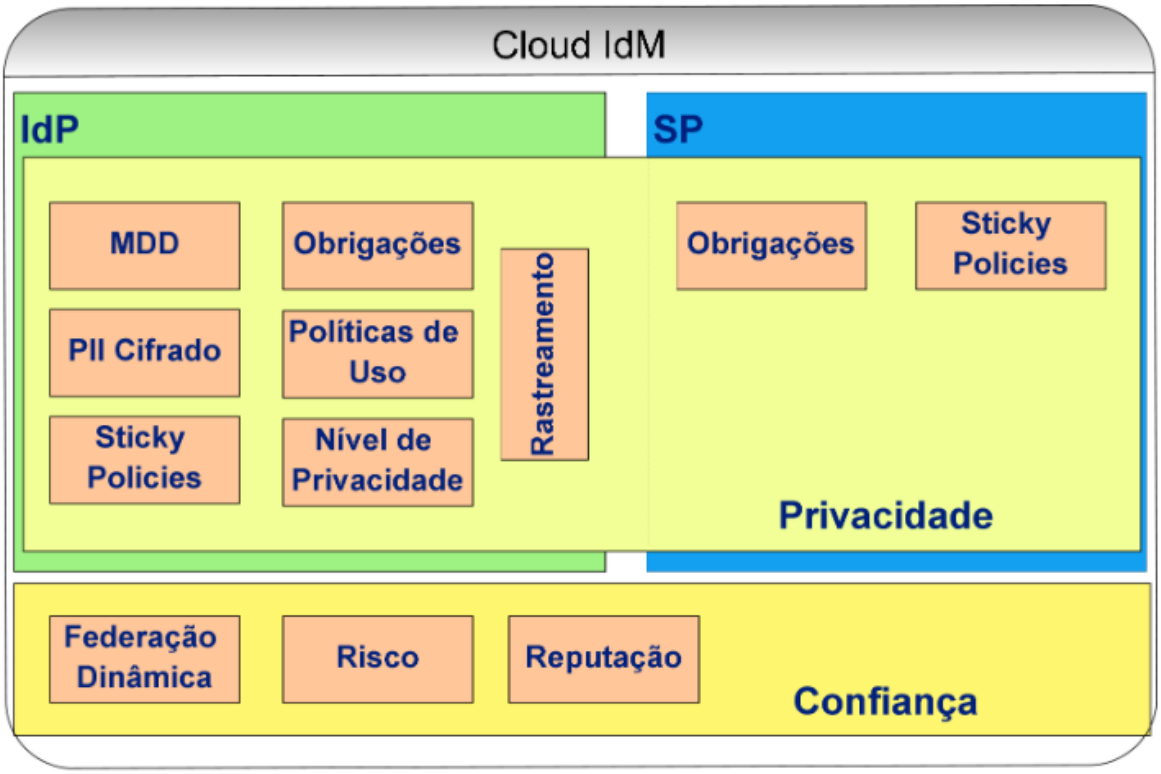

Figura 11 – Modelo de IdM com privacidade em nuvem (Werner, 2017, p. 90)

4 PROPOSTA: POLÍTICA DE SINCRONIZAÇÃO COM IDP

Esta proposta introduz o conceito de política de sincronização com IdP (ou *IdP Sync Policy*) ao ACS, que é a formalização de como o sistema deve interpretar os atributos recebidos do IdP. Isto é, uma política de sincronização é um conjunto que reúne: um *script* de mapeamento dos dados recebidos; a URL do IdP que forneceu os dados; e o tipo de operação a ser realizada com estes dados.

A Figura 12 descreve o processo atualmente executado pelo ACS para validação da resposta recebida do IdP após o processo de autenticação do usuário. Ele procura pelo atributo com determinado nome (previamente configurado pelo administrador em ambos ACS e IdP) e realiza o *login* do primeiro usuário encontrado cujo *username* seja igual ao valor do atributo. Uma vez logado, o usuário pode trocar entre contas com mesmo *username* através de outro *endpoint* da API.

Apesar de não relevantes para a política de sincronização proposta, estão presentes na Figura o *token* para identificação da requisição à qual a resposta se refere e o `SAML NameID`, que é repassado no envio do comando de SLO.

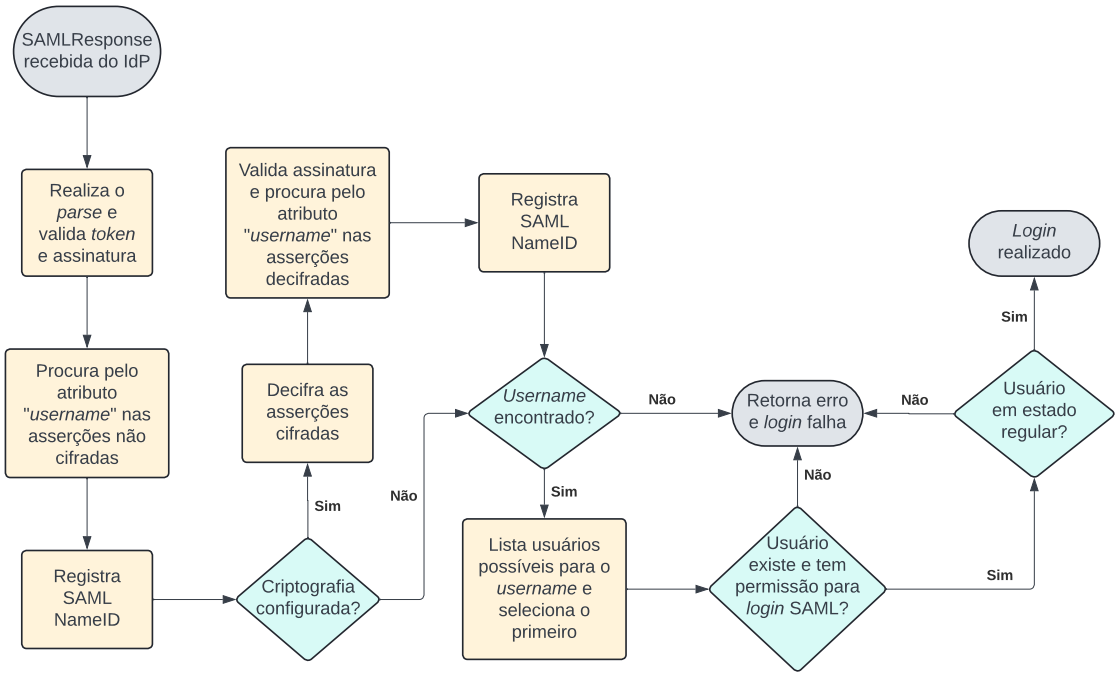

Figura 12 – Fluxograma do *login* SAML presente no ACS.

Em contraste, o fluxo planejado para a nossa implementação se encontra na Figura 13. Ao invés de procurar apenas o atributo contendo o *username*, todos os atributos serão repassados ao controlador da política de sincronização. A seção "Agregação dos atributos SAML" é especifica ao tratamento do protocolo SAML, de acordo com o escopo do trabalho, mas a parte de mapeamento e execução das políticas de acordo

com a operação é agnóstica e pode ser aplicada a atributos recebidos por quaisquer protocolos, sendo necessário estender as respectivas implementações no ACS para que realizem esta agregação.

Os atributos são injetados no interpretador JavaScript, que executa o *script* e retorna um objeto que representa um usuário no contexto do ACS. Se o usuário equivalente não existir no banco de dados de metadados do ACS e a política permitir, ele é criado. Se houver informações conflitantes com o usuário existente e a política permitir, ele é atualizado.

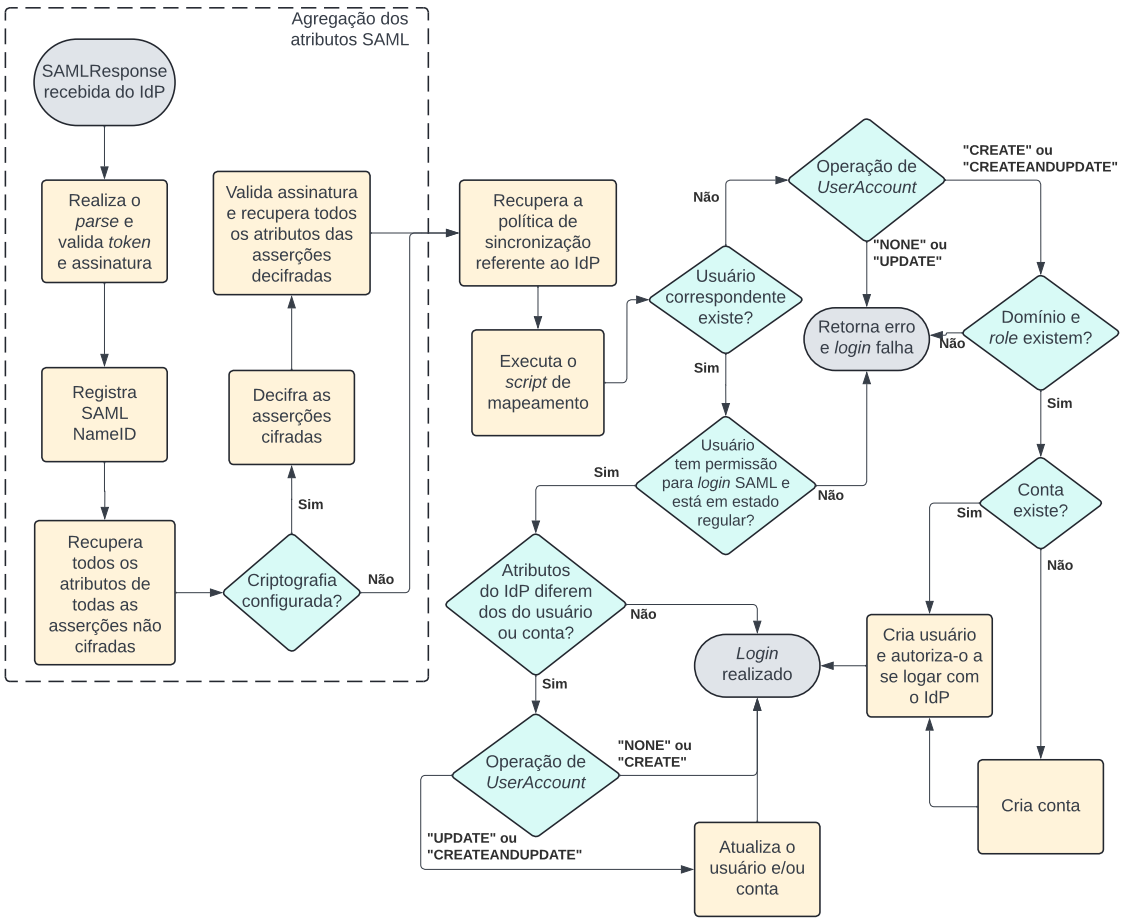

Figura 13 – Fluxograma proposto para o *login* SAML do ACS.

4.1 OPERAÇÃO

Nesta primeira versão, o único tipo de operação presente será o de *UserAccounts* (termo que o ACS utiliza para se referir ao conjunto de usuário e conta). A operação pode ser uma das seguintes:

- `NONE`: Rejeita o *login* se o usuário não for identificado e ignora diferenças nos atributos;

- `CREATE`: Cria usuário e/ou conta se não forem identificados;
- `UPDATE`: Rejeita o *login* se o usuário não for identificado. Em caso de contradições, atualiza usuário e/ou conta para refletir os atributos recebidos do IdP;
- `CREATEANDUPDATE`: Cria usuário e/ou conta se não forem identificados. Em caso de contradições, atualiza usuário e/ou conta para refletir os atributos recebidos do IdP.

Em extensões futuras, mais tipos de operação podem ser adicionados para permitir a flexibilização da criação de domínios e *roles*, por exemplo. Todavia, devido à natureza de *login* desta comunicação com o IdP, a operação de *UserAccounts* servirá como escopo para os outros tipos de operação (por exemplo, não deverá ser possível criar um domínio sem criar uma conta e usuário).

4.2 MAPEAMENTO

Desde a versão 4.18, lançada em 2023, o ACS permite definir a contabilização de recursos por *scripts* JavaScript com variáveis de ambiente pré-estabelecidas, chamados de regras de ativação[1]. O mesmo interpretador é também utilizado em seletores de armazenamento desde a versão 4.19[2].

Este trabalho reaproveita a funcionalidade para que o operador realize o mapeamento dos atributos recebidos do IdP (injetados no *script* através do objeto "`idp`") para uma estrutura pré-definida, representada por um objeto JavaScript que deve ser retornado ao fim da execução.

Este objeto retornado deve conter determinados atributos, especificados na lista a seguir. Dependendo das operações definidas, diferentes combinações de atributos podem ser obrigatórias. No caso de conflitos, o atributo "`uuid`" terá precedência sobre outros identificadores de um mesmo objeto.

- `user`:
  - `email`: Obrigatório para criação de usuário/conta.
  - `firstname`: Obrigatório para criação de usuário/conta.
  - `lastname`: Obrigatório para criação de usuário/conta.
  - `username`: Obrigatório para criação de usuário/conta e para *login*.
  - `uuid`: Opcional. Se presente, todos os outros atributos, incluídos os de `account` e `domain`, deixam de ser obrigatórios para *login*.

[1] https://github.com/apache/cloudstack/pull/5909 (Quota custom tariffs)

[2] https://github.com/apache/cloudstack/pull/7659 (Add dynamic secondary storage selection)

  - `timezone`: Opcional.

- `account`:
  - `accountname`: Opcional.
  - `details`: Opcional. Os atributos são no formato de nome da configuração e o respectivo valor.
    * `config.exemplo`: valor
    * ...
  - `networkdomain`: Opcional.
  - `role`:
    * `name`: Obrigatório para criação de conta.
    * `type`: Obrigatório para criação de conta se houver mais de uma *role* com o mesmo `name`.
    * `uuid`: Opcional. Se presente, `name` e `type` deixam de ser obrigatórios para criação de conta.
  - `uuid`: Opcional. Se presente e a conta existir, `domain` deixa de ser obrigatório para *login* e criação de usuário.

- `domain`:
  - `path`: Obrigatório para criação de usuário/conta e para *login*.
  - `uuid`: Opcional. Se presente, `path` deixa de ser obrigatório.

- `legacy`: Opcional. Se presente, indica que a política deve tentar realizar o *login* de um mapeamento que retorne apenas "`user.username`".

Um exemplo é provido no Código 1. Neste caso, o ACS tentará realizar o *login* de um usuário com: *username* igual a uma combinação dos atributos recebidos do IdP `firstName` e `lastName` (linha 6); e domínio igual a "`/federated/fln`", "`/federated/sp`" ou "`/federated/others`" (linhas 16-28), de acordo com o valor do atributo recebido do IdP `areacode` (linha 19).

Se encontrado o usuário e a operação permitir, ele poderá ter seus atributos do ACS atualizados. Da mesma forma, se não encontrado e a operação permitir, ele poderá ser criado.

Código 1 – Exemplo de mapeamento de política de sincronização

```
res = {
    user: {
        email: idp["e-mail"],
        firstname: idp["firstName"],
        lastname: idp["lastName"],
        username: idp.firstName[0] + idp.lastName
    }
}
res.account = {
    accountname: idp.firstName + "-" + idp["Office Branch"],
    role: {
        name: idp.manager ? "DomainAdmin" : idp.title,
        type: idp.manager ? "DomainAdmin" : "User"
    }
}
res.domain = {
    path: "/federated/"
}
switch (idp.areacode) {
    case 48:
        res.domain.path += "fln"
        break
    case 11:
        res.domain.path += "sp"
        break
    default:
        res.domain.path += "others"
}
res
```

Excepcionalmente, o mapeamento pode conter apenas o atributo `user.username`. Apesar de manter a ambiguidade de domínio (que não existe em qualquer outro lugar do ACS), a decisão foi tomada para manter a compatibilidade com o comportamento já implementado na plataforma.

4.3 BANCO DE DADOS

No banco de dados, a tabela `idp_sync_policy` armazenará a política de sincronização, com as colunas descritas na Tabela 2.

Coluna	Tipo	Descrição
`id`	`bigint(20) unsigned`	Chave primária, ID interno.
`uuid`	`varchar(40)`	Chave única, UUID externo.
`idp_id`	`text`	URL do IdP.
`mapping`	`text`	*Script* de mapeamento dos atributos recebidos do IdP aos atributos da entidade do ACS.
`useraccount_operation`	`enum('NONE', 'CREATE', 'UPDATE', 'CREATEANDUPDATE')`	Operação a ser realizada sobre usuário e conta. Valor padrão é `"NONE"`.
`description`	`text`	Descrição opcional.
`created`	`datetime`	*Timestamp* de criação.
`updated`	`datetime`	*Timestamp* da última atualização.
`update_count`	`bigint(20) unsigned`	Número de atualizações.
`removed`	`datetime`	*Timestamp* de remoção.
`removal_reason`	`text`	Motivo para remoção.

Tabela 2 – Colunas da tabela `idp_sync_policy`

4.4 API

Para modificar e retornar a tabela criada, serão criados os *endpoints* da API `createIdpSyncPolicy`, `updateIdpSyncPolicy`, `removeIdpSyncPolicy` e `listIdpSyncPolicies`. Seus parâmetros estão dispostos nas tabelas 3, 4, 5 e 6, respectivamente. Todos os comandos produzirão eventos para o histórico do ACS.

A chamada `updateIdpSyncPolicy` não permite a troca de operações a fim de preservar o histórico de alterações e facilitar processos de auditoria, uma vez que a alteração da operação pode causar efeitos significativos. Para realizar esta modificação, basta recriar a política de sincronização.

Parâmetro	Tipo	O	VP	Descrição
`idpid`	*String*	Sim	—	IdP ao qual a política se aplicará.
`description`	*String*	Não	—	Descrição para controle do administrador.
`mapping`	*String*	Sim	—	*Script* de mapeamento.
`useraccountoperation`	*String*	Não	`"NONE"`	Operação de usuário e conta.

O = Obrigatório; VP = Valor Padrão.

Tabela 3 – Parâmetros de `createIdpSyncPolicy`

Parâmetro	Tipo	O	VP	Descrição
`id`	*String*	Sim	—	Política de sincronização a ser atualizada.
`description`	*String*	Não	—	Descrição para substituir a anterior.
`mapping`	*String*	Não	—	*Script* para substituir o anterior.

O = Obrigatório; VP = Valor Padrão.

Tabela 4 – Parâmetros de `updateIdpSyncPolicy`

Parâmetro	Tipo	O	VP	Descrição
`id`	*String*	Sim	—	Política de sincronização a ser removida.
`removalreason`	*String*	Sim	—	Motivo para a remoção da política.

O = Obrigatório; VP = Valor Padrão.

Tabela 5 – Parâmetros de `removeIdpSyncPolicy`

Parâmetro	Tipo	O	VP	Descrição
`id`	*String*	Não	—	Política de sincronização a ser listada.
`idpid`	*String*	Não	—	IdP da política de sincronização a ser listada.
`keyword`	*String*	Não	—	Palavra-chave presente na descrição.
`showremoved`	*Boolean*	Não	Falso	Se a resposta deve incluir políticas removidas.

O = Obrigatório; VP = Valor Padrão.

Tabela 6 – Parâmetros de `listIdpSyncPolicies`

Além destes, também será implementado um *endpoint* chamado `validateIdpSyncPolicyMapping`, que recebe os parâmetros descritos na Tabela 7 e responde com o objeto retornado em uma simulação do processo de *login*, sem realizar alterações no banco de dados.

Parâmetro	Tipo	O	VP	Descrição
`idp`	*String*	Sim	—	Objeto JSON que simule os atributos recebidos do IdP.
`mapping`	*String*	Sim	—	*Script* de mapeamento.
`useraccountoperation`	*String*	Não	`"NONE"`	Operação de usuário e conta.

O = Obrigatório; VP = Valor Padrão.

Tabela 7 – Parâmetros de `validateIdpSyncPolicyMapping`

4.5 CONFIGURAÇÕES

Será adicionada a configuração global `idp.sync.policy.mapping.timeout`, que determina o tempo máximo de espera pela conclusão do processamento do mapeamento de uma política de sincronização.

A configuração `saml2.user.attribute` será marcada como deprecada e removida em versões posteriores. Para manter o comportamento atual, existirá o atributo `legacy` no *script* de mapeamento que, quando verdadeiro, permitirá que o mapeamento retorne um objeto contendo apenas *username*, preservando o *login* da implementação atual.

Além disso, será criada a configuração `idp.sync.policy.auto.create.default.policy`, que habilita a criação automática de uma política de sincronização padrão quando encontrado um novo IdP, com o *script* de mapeamento do Código 2 e operação de *UserAccount* de tipo `NONE`.

Código 2 – *Script* de mapeamento das políticas padrão.

```
1 r = { "user": { "username": idp.<valor de saml2.user.attribute> },
    "legacy": true }
```

5 IMPLEMENTAÇÃO

O código desenvolvido pode ser encontrado no Apêndice A. De acordo com a forma que o ACS é estruturado, foram criadas as representações `IdpSyncPolicy` e `IdpSyncPolicyVO`, bem como o objeto de acesso (*Data Access Object* – DAO) `IdpSyncPolicyDAO` para abstração do relacionamento com o banco de dados referente às políticas de sincronização.

Além destes, foi estabelecido o componente `IdpSyncPolicyManager`, que é chamado pelo *plugin* SAML depois de processar a resposta SAML e coletar os atributos enviados pelo IdP, para executar a política apropriada.

Foram criadas algumas classes intermediárias (`AttributeMap`, `UserMap`, `AccountMap`, `DomainMap`, `RoleMap`) para modelar os atributos a serem recebidos. A principal desvantagem desta abordagem de acoplamento é que, quando alteradas as classes de usuário, domínio, conta, *role*, etc., pode ser necessário também alterar estas classes de mapeamento.

Em contrapartida, o benefício é encontrado no isolamento do funcionamento interno do banco do ACS, reutilizando as mesmas checagens que ocorrem no fluxo mais comum de operações através dos *endpoints* diretos, como *createAccount*, *updateUser*, etc. Assim, não apenas os valores dos atributos são validados, como também o estado do usuário dentro do ACS (se está desabilitado ou bloqueado, por exemplo) também é respeitado.

Para realizar a adaptação para o novo modelo, foi adicionada uma verificação à inicialização do *plugin* SAML. Ela ocorre durante a análise dos metadados dos IdPs, se a configuração `idp.sync.policy.auto.create.policy` for verdadeira. Para cada IdP adicionado, o ACS procura por uma política de sincronização ativa daquele IdP. Se não encontrada, uma padrão é criada, como descrito em 4.5.

5.1 CONFIGURAÇÃO DO AMBIENTE

Os testes foram realizados em um ambiente com o ACS (*release candidate* da versão 4.20[1], com as alterações deste trabalho) rodando em uma máquina virtual e o Keycloak executado a partir de uma imagem Docker[2].

Na instância do Keycloak, foram criados quatro usuários *userpadrao*, *usera*, *userb* e *userc* e importados os dados de SP do ACS obtidos pelo comando `getSPMetadata`. Dessa forma, foi possível definir alguns *mappers*, apresentados na Figura 14, que descrevem quais informações dos usuários o Keycloak deve prover ao SP.

No ACS, a configuração `saml2.idp.metadata.url` recebeu o valor da URL provida

[1] https://github.com/apache/cloudstack/tree/4.20.0.0-RC20241021T1

[2] https://www.keycloak.org/getting-started/getting-started-docker

pelo Keycloak.

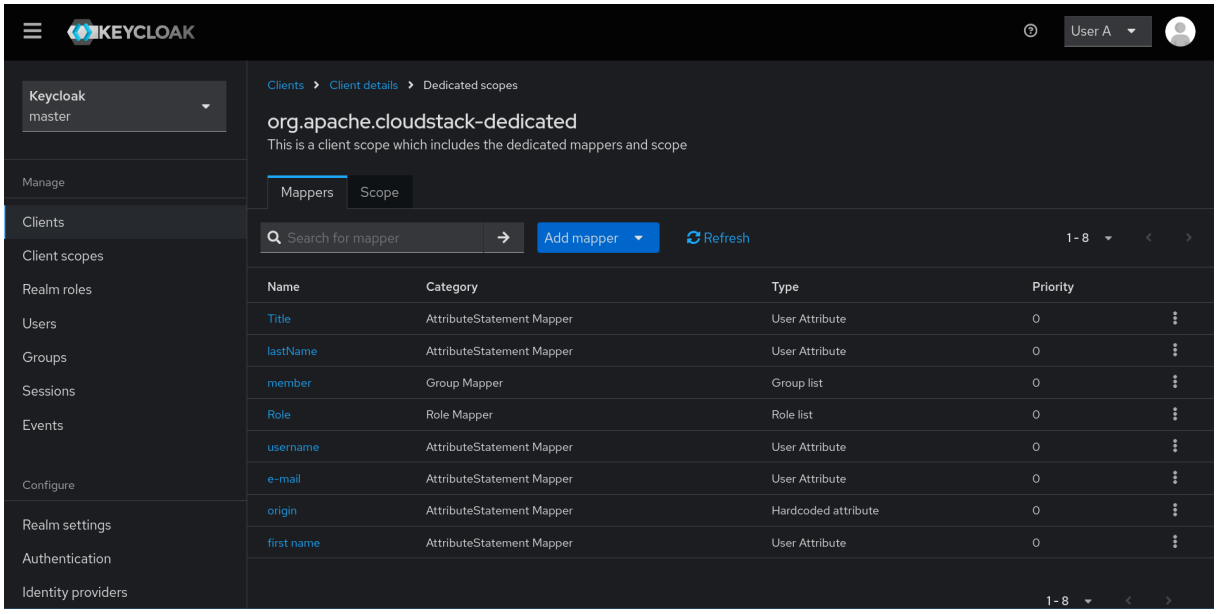

Figura 14 – *Mappers* do Keycloak no ambiente de teste.

5.2 API

Foram criadas as classes referentes aos comandos descritos em 4.4 (`CreateIdpSyncPolicyCmd`, `UpdateIdpSyncPolicyCmd`, `RemoveIdpSyncPolicyCmd`, `ListIdpSyncPoliciesCmd` e `ValidateIdpSyncPolicyMappingCmd`), que definem os parâmetros a serem recebidos na chamada de API e chamam o gerenciador. Os comandos só serão acessíveis e adicionados à lista de comandos se o *plugin* SAML2 estiver habilitado (`saml2.enabled`).

Utilizando o utilitário CloudMonkey, a Figura 15 apresenta o envio de uma requisição ao *endpoint* `createIdpSyncPolicy`, com os parâmetros descritos na Tabela 3. A operação de *UserAccount* da política de sincronização criada é de tipo `CREATEANDUPDATE`; o seu *script* de mapeamento está no Código 3; o IdP refere-se ao servidor Keycloak de teste `http://192.168.123.1:8080/auth/realms/master`; e a descrição é apenas "`Test policy`".

O ACS então retorna a política de sincronização que acaba de ser criada, com os atributos não nulos descritos na seção 4.3: o *timestamp* de criação, a descrição, o UUID da política de sincronização, o ID do IdP, o *script* de mapeamento e a operação de *UserAccount*.

```
(local) 🐱 > create idpsyncpolicy useraccount_operation=CREATEANDUPDATE mapping="r = {user: { username: idp.username, firstname: idp[\"first name\"], lastname: idp.lastName, email: idp[\"e-mail\"] }, domain: { p
ath: \"/\"}, account: { role: { name: idp.member[0]} } }" idpid=http://192.168.123.1:8080/auth/realms/master description="Test policy"
{
  "idpsyncpolicy": {
    "created": "2024-11-18T11:48:38+0000",
    "description": "Test policy",
    "id": "ef012193-7434-4f39-9e6d-5081749a1559",
    "idpid": "http://192.168.123.1:8080/auth/realms/master",
    "mapping": "r = {user: { username: idp.username, firstname: idp[\"first name\"], lastname: idp.lastName, email: idp[\"e-mail\"] }, domain: { path: \"/\"}, account: { role: { name: idp.member[0]} } }",
    "useraccount_operation": "CREATEANDUPDATE"
  }
}
```

Figura 15 – `createIdpSyncPolicy` pelo CloudMonkey

Código 3 – *Script* de mapeamento da Figura 15.

```
r = {
    user: {
        username: idp.username,
        firstname: idp["first name"],
        lastname: idp.lastName,
        email: idp["e-mail"]
    },
    domain: {
        path: "/"
    },
    account: {
        role: {
            name: idp.member[0]
        }
    }
}
```

Da mesma forma, a Figura 16 apresenta a utilização do comando `updateIdpSyncPolicy` para alteração da descrição para "`Updated`"; a Figura 17, a remoção da política de sincronização pelo comando `removeIdpSyncPolicy`, com o motivo "`Testing API`"; e a Figura 18, a listagem da política removida, com o comando `listIdpSyncPolicies`.

Figura 16 – `updateIdpSyncPolicy` pelo CloudMonkey

Figura 17 – `removeIdpSyncPolicy` pelo CloudMonkey

Figura 18 – `listIdpSyncPolicies` pelo CloudMonkey

Ainda, os comandos (à exceção do `validateIdpSyncPolicyMapping`) geram eventos, que são guardados na tabela `event` do banco de dados e apresentados na sua própria seção da UI, ilustrada na Figura 19. A criação da política de sincronização padrão também é registrada como um evento, normalmente.

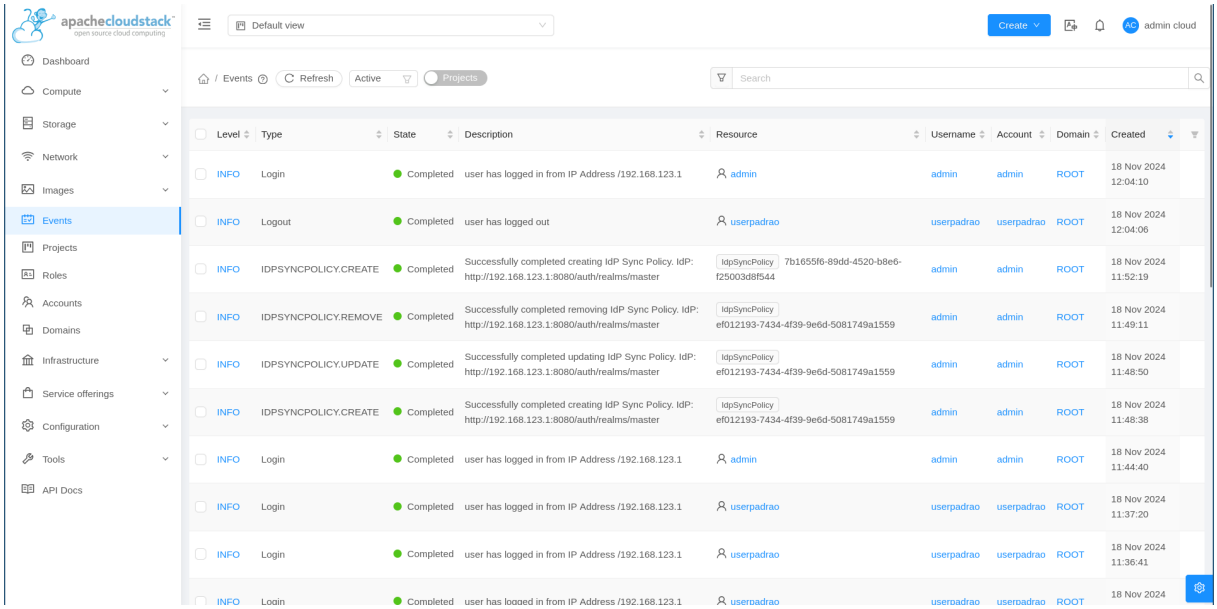

Figura 19 – Página de eventos do ACS

O comando `validateIdpSyncPolicyMapping` segue o mesmo fluxo que o processo de *login*. Este fluxo diverge no momento de realizar a criação e atualização das entidades, que retorna uma mensagem ao invés de persistir as mudanças no banco de dados.

A Figura 20 apresenta a validação do *script* do Código 3 com os atributos "`Title`, `lastName`, `member`, `username`, `e-mail`, `origin` e `first name`" e operação de *UserAccount* de tipo `CREATEANDUPDATE`.

Já que o usuário resolvido pelo *script* de mapeamento não existe e a operação permite a criação, o ACS responde que tentaria criar a conta com os atributos mapeados presentes no campo "`mappedattributes`" da resposta.

```
(local) 🐱 > validate idpsyncpolicymapping useraccount_operation=CREATEANDUPDATE idp="{Title: \"Consultant\", lastName: \"Test Lastname\", member: [\"Contractor\"], username: \"userteste\", \"e-mail\": \"e-mail@place.com\", origin: \"keycloak\", \"first name\": \"Test FirstName\" }" mapping="r = {user: { username: idp.username, firstname: idp[\"first name\"], lastname: idp.lastName, email: idp[\"e-mail\"] }, domain: { path: \"/\"}, account: { role: { name: idp.member[0]} } }"

{
  "displaytext": "Would try to create UserAccount in domain [/].",
  "mappedattributes": {
    "account": {
      "role": {
        "name": "Contractor"
      }
    },
    "domain": {
      "path": "/"
    },
    "user": {
      "email": "e-mail@place.com",
      "firstname": "Test FirstName",
      "lastname": "Test Lastname",
      "username": "userteste"
    }
  },
  "success": true
}
```

Figura 20 – `validateIdpSyncPolicyMapping` pelo CloudMonkey

5.3 *LOGIN*

O comando de *login SAML*, `SAML2LoginAPIAuthenticatorCmd`, foi modificado para coletar todos os atributos recebidos ao invés de procurar apenas o nome de usuário descrito na configuração `saml2.user.attribute`. Uma vez coletados, eles são encaminhados ao gerenciador das políticas de sincronização, `IdpSyncPolicyManager`.

Por sua vez, o gerenciador de políticas executa o fluxo da Figura 13, exceto se o *script* de mapeamento retornar apenas os atributos "`username`" e "`legacy`", quando apresenta o comportamento antigo (Figura 12).

5.3.1 Atualização de conta e/ou usuário

Se o usuário for encontrado, o gerenciador de políticas verifica se a operação permite e, em seguida, compara os atributos recebidos com os de usuário e conta presentes no ACS. Se algum deles for diferente, delega ao gerenciador de conta a execução da alteração.

A Figura 21 contém o *log* gerado pelo gerenciador de política de sincronização logo após receber os atributos do módulo de autenticação SAML. Na Figura 22, o *script* de mapeamento é executado e retorna os atributos relevantes. Na Figura 23, é avisado que o campo referente à senha não é considerado e procura-se pelo usuário `userb` no domínio de ID `1`.

```
2024-11-18 13:38:47,776 DEBUG [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Attributes received from IdP
 [http://192.168.123.1:8080/auth/realms/master] are [{"Role":"manage-account-links","lastName":"B","first name":"User","origin":"keycloak","member
":"/Contractor","Title":"Consultant","e-mail":"novoemaildouserb@email.com","username":"userb"}].
```

Figura 21 – *Log* do recebimento dos atributos pelo gerenciador de políticas de sincronização.

```
2024-11-18 13:38:47,782 DEBUG [o.a.c.u.j.JsInterpreter] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Executing script [ idp = {"Role":"manag
e-account-links","lastName":"B","first name":"User","origin":"keycloak","member":"/Contractor","Title":"Consultant","e-mail":"novoemaildouserb@ema
il.com","username":"userb"}; r = {user: { username: idp.username, firstname: idp["first name"], lastname: idp.lastName, email: idp["e-mail"] }, do
main: { path: "/"}, account: { role: { name: ("member" in idp) ? idp.member.substring(1) : idp.Title } } };].
2024-11-18 13:38:47,818 DEBUG [o.a.c.u.j.JsInterpreter] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) The script [ idp = {"Role":"manage-acco
unt-links","lastName":"B","first name":"User","origin":"keycloak","member":"/Contractor","Title":"Consultant","e-mail":"novoemaildouserb@email.com
","username":"userb"}; r = {user: { username: idp.username, firstname: idp["first name"], lastname: idp.lastName, email: idp["e-mail"] }, domain:
{ path: "/"}, account: { role: { name: ("member" in idp) ? idp.member.substring(1) : idp.Title } } };] had the following result: [[object Object]]
.
2024-11-18 13:38:47,819 TRACE [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Attributes resolved from scr
ipt: [{"user":{"username":"userb","firstname":"User","lastname":"B","email":"novoemaildouserb@email.com"},"domain":{"path":"/"},"account":{"role":
{"name":"Contractor"}}}]
2024-11-18 13:38:47,819 DEBUG [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Mapped attributes: [{"user":
{"email":"novoemaildouserb@email.com","firstname":"User","lastname":"B","username":"userb"},"account":{"role":{"name":"Contractor"}},"domain":{"pa
th":"/"}}]
```

Figura 22 – *Logs* da execução do *script* de mapeamento.

```
2024-11-18 13:38:47,819 WARN  [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Password field found, but it
 is not expected and will not be used.
2024-11-18 13:38:47,819 TRACE [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Searching for domain [/].
2024-11-18 13:38:47,821 TRACE [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-25:[ctx-cbb2309b]) (logid:29eec61f) Searching for user with user
name [userb] and domain [1].
```

Figura 23 – *Logs* da procura do usuário pelo gerenciador de políticas de sincronização.

Em seguida, na Figura 24, verifica-se que há diferenças entre os atributos recebidos e o usuário encontrado tem o seu e-mail atualizado para `novoemaildouserb@email.com`. Por fim, a Figura 25 apresenta o retorno da política de sincronização, que encaminha o usuário para ser logado.

Figura 24 – *Logs* da atualização do usuário no gerenciador de políticas de sincronização.

Figura 25 – *Logs* do retorno do usuário e *login* após atualização.

5.3.2 Criação de conta e/ou usuário

Se uma conta for encontrada, mas não um usuário, o gerenciador de políticas verifica se a operação permite a criação e se os atributos necessários estão presentes. Em caso positivo, chama o gerenciador de contas, que realizará a criação de um novo usuário na conta definida.

Se não encontrada a conta, o processo é semelhante: os atributos e a operação são verificados e o gerenciador de contas é chamado para criar ambos conta e usuário.

Na Figura 26 os mesmos passos são realizados que os presentes nas Figuras 21, 22 e 23. O processo diferencia-se na Figura 27, onde é chamado o gerenciador de contas, que se encarrega de realizar a criação da conta e do usuário. Por fim, a Figura 28 novamente retorna o usuário e realiza o *login*.

Figura 26 – *Logs* do recebimento de atributos, execução do *script* de mapeamento e procura do usuário.

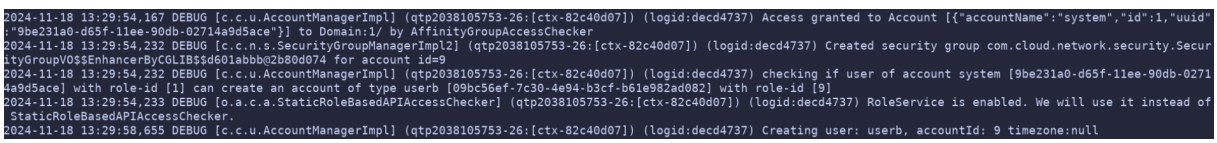

Figura 27 – *Logs* da criação de conta e usuário.

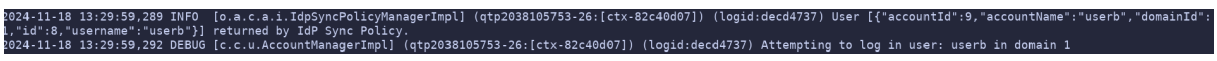

Figura 28 – *Logs* do retorno do usuário e *login* após criação.

A Figura 29 contém a lista de contas presente no ambiente (acessada como administrador) antes do teste de *login* com criação.

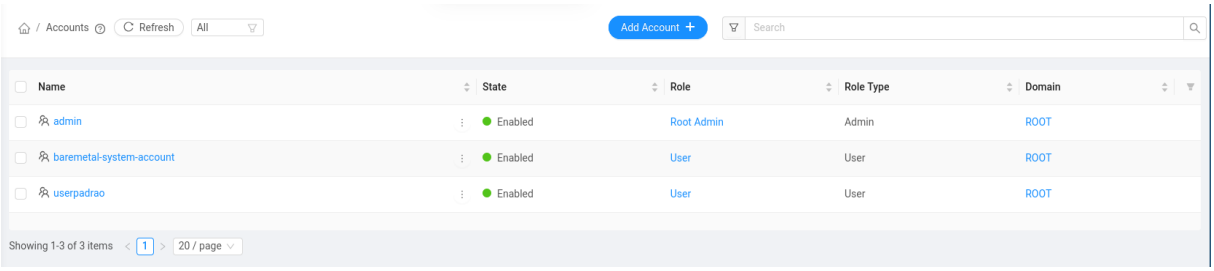

Figura 29 – Lista de contas no ambiente de teste.

As Figuras 30, 31 e 32 ilustram os passos envolvidos no *login*, respectivamente: a tela de SSO do ACS, a tela de *login* no Keycloak ao qual o usuário é direcionado e a tela logada no ACS, após redirecionamento do Keycloak.

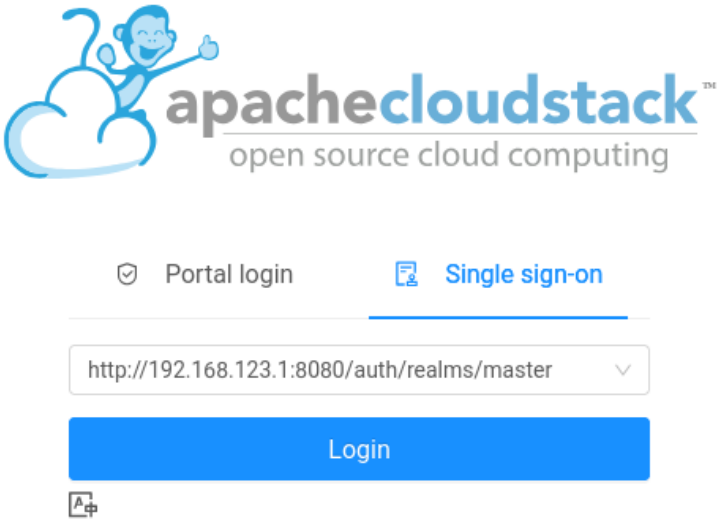

Figura 30 – Tela de SSO no ACS.

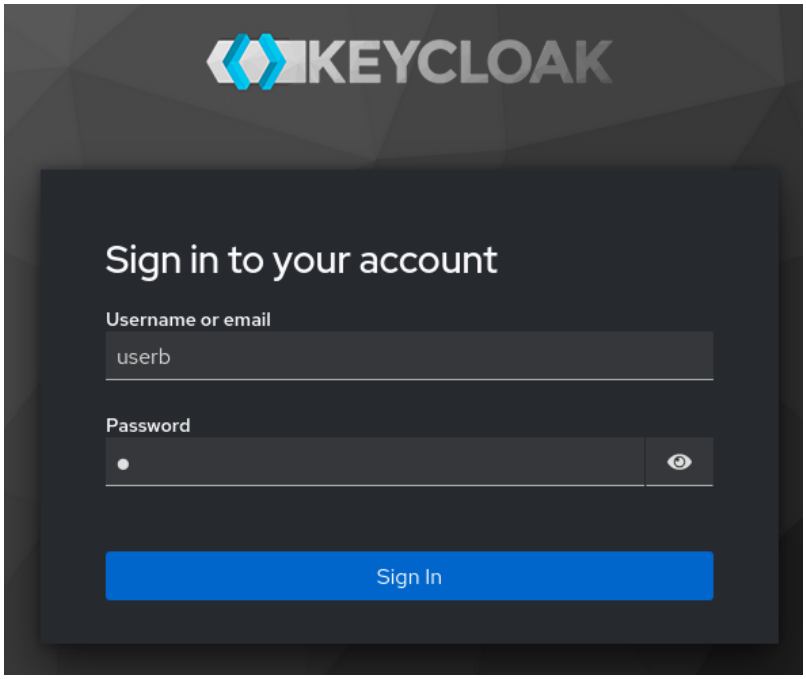

Figura 31 – Tela de *login* do Keycloak.

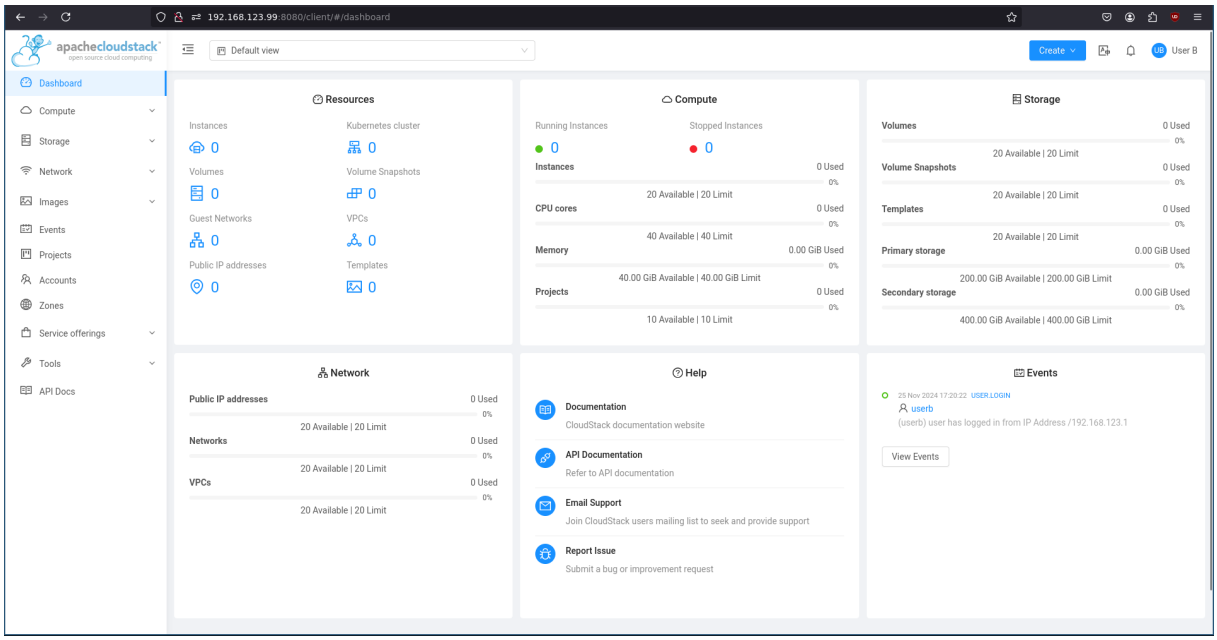

Figura 32 – *Dashboard* do ACS, logado como `userb`.

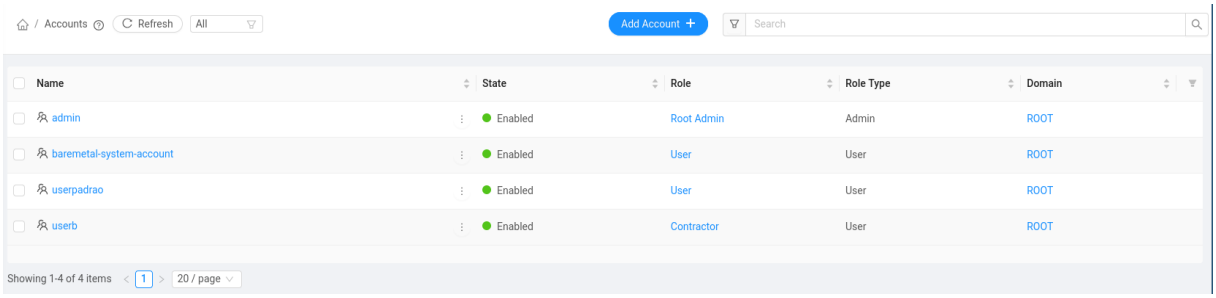

Figura 33 – Lista de contas no ambiente de teste com a conta recém criada.

A Figura 33 apresenta a lista de contas, com a conta que acabou de ser criada.

5.3.3 Política de sincronização padrão

Quando uma política de sincronização retorna apenas os atributos "`username`" e "`legacy`" (este último, com valor verdadeiro), um *bypass* do processo normal de verificação é realizado para manter a compatibilidade com o comportamento atual do ACS. Neste caso, o primeiro usuário encontrado com o *username* e que esteja autorizado a se logar pelo IdP é escolhido.

No processo de *login* cujos *logs* estão presentes na Figura 34, o ACS reúne os atributos recebidos do IdP (os mesmos definidos nos *mappers* da Figura 14): "`Role`", "`lastName`", "`first name`", "`origin`", "`Title`", "`e-mail`" e "`username`". Em seguida, executa o *script* e identifica que existe apenas o atributo "`username`" com valor não nulo. Portanto, realiza o *bypass* das outras verificações e se loga no primeiro usuário encontrado que esteja autorizado a se logar com SAML no IdP e que tenha o *username* "`userpadrao`".

```
2024-11-18 11:37:20,358 DEBUG [o.a.c.a.c.SAML2LoginAPIAuthenticatorCmd] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1
838) Received SAMLResponse in response to id=flonvlp9bbv8hauuf570nrg2fsljiv3h
2024-11-18 11:37:20,382 DEBUG [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838)
Attributes received from IdP [http://192.168.123.1:8080/auth/realms/master] are [{"Role":"manage-account-links","lastN
ame":"","first name":"","origin":"keycloak","Title":";","e-mail":"","username":"userpadrao"}].
2024-11-18 11:37:20,419 DEBUG [o.a.c.u.j.JsInterpreter] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) Executing s
cript [ idp = {"Role":"manage-account-links","lastName":"","first name":"","origin":"keycloak","Title":";","e-mail":""
,"username":"userpadrao"}; r = {"user": { "username": idp.username }, "legacy": true }].
2024-11-18 11:37:20,490 DEBUG [o.a.c.u.j.JsInterpreter] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) The script
[ idp = {"Role":"manage-account-links","lastName":"","first name":"","origin":"keycloak","Title":";","e-mail":"","user
name":"userpadrao"}; r = {"user": { "username": idp.username }, "legacy": true }] had the following result: [[object O
bject]].
2024-11-18 11:37:20,491 TRACE [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838)
Attributes resolved from script: [{"user":{"username":"userpadrao"},"legacy":true}]
2024-11-18 11:37:20,492 DEBUG [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838)
Mapped attributes: [{"user":{"username":"userpadrao"},"legacy":true}]
2024-11-18 11:37:20,492 WARN  [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838)
Bypassing usual flow due to policy mapping containing a single "username" attribute as well as 'legacy' being true.
2024-11-18 11:37:20,495 INFO  [o.a.c.a.i.IdpSyncPolicyManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838)
User [{"accountId":8,"accountName":"userpadrao","domainId":1,"id":7,"username":"userpadrao"}] returned by IdP Sync Pol
icy.
2024-11-18 11:37:20,499 DEBUG [c.c.u.AccountManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) Attempting
 to log in user: userpadrao in domain 1
2024-11-18 11:37:20,509 DEBUG [o.a.c.s.SAML2UserAuthenticator] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) Tryi
ng SAML2 auth for user [userpadrao].
2024-11-18 11:37:20,530 DEBUG [c.c.u.AccountManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) CIDRs from
 which account 'Account [{"accountName":"userpadrao","id":8,"uuid":"ca0b1dbe-ef01-4ce1-ba34-d70a31b1b114"}]' is allowe
d to perform API calls: 0.0.0.0/0,::/0
2024-11-18 11:37:20,532 DEBUG [c.c.u.AccountManagerImpl] (qtp2038105753-23:[ctx-462ef3e2]) (logid:e5fa1838) User: user
padrao in domain 1 has successfully logged in
```

Figura 34 – *Logs* do *login* com uma política de sincronização padrão.

6 CONCLUSÃO

O Apache CloudStack é uma plataforma de orquestração em nuvem com diferentes opções para controle de acesso, mas todas limitadas no âmbito de atributos. A utilização de atributos no processo de controle de acesso proporciona maior flexibilidade e escalabilidade em sistemas em rede.

Foram contextualizados conceitos da literatura relacionados aos temas de gestão de identidade e computação em nuvem e diagramado o fluxo atual de comunicação SAML do ACS. A partir destes, propusemos uma abordagem denominada em políticas de sincronização.

Dado este entendimento e a fim de iniciar uma discussão na comunidade do ACS em torno dos temas de autenticação e autorização, a abordagem foi implementada para possibilitar que o ACS delegue a gerência das identidades do sistema, criando e atualizando as suas identidades quando deparado com informações novas vindas do provedor de identidade.

Apesar dos limites presentes no escopo atual da implementação, o trabalho espera abrir caminho para futuras inovações e melhorias no processo de integração entre o Apache CloudStack e os provedores de identidade, sugerindo, a seguir, novas possibilidades para a evolução da plataforma.

6.1 TRABALHOS FUTUROS

Algumas das melhorias previstas em volta da funcionalidade apresentada neste trabalho incluem:

- Integração com outros protocolos de SSO, como OAuth e OIDC;
- Gerenciamento das políticas de sincronização pela UI;
- Restrição de criação e atualização de contas por IP;
- Criação e/ou atualização automática de *roles*, domínios e projetos;
- Restrição de permissões de acordo com os atributos recebidos, tal qual RBAC-A com foco em *roles* descrito por Kuhn, Coyne e Weil (2010);
- Agregação de atributos de múltiplos IdPs, como proposto por Chadwick e Inman (2009);
- Flexibilização ou restrição das permissões de acesso aos recursos em si, indo além de apenas a existência ou não de permissão de acesso ao *endpoint* da API;
- Restringir e/ou remover recursos de acordo com as informações do IdP.

Uma funcionalidade de alta complexidade, mas que poderia trazer diversos benefícios é possibilitar que o ACS atue como um IdP. Além de permitir a participação em federações de identidade, também resultaria em melhor integração de ambientes distribuídos em diferentes regiões, permitindo a centralização da gestão de identidades sem a necessidade de *software* terceiro.

Outra importante possibilidade de trabalho futuro seria um modo de proporcionar controle ao usuário sobre seus atributos quando recebidos pelo SP (neste caso, o ACS), como sugerido por Weingärtner (2014).

REFERÊNCIAS

ABADI, Martín. Logic in Access Control (Tutorial Notes). In: **Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures**. Edição: Alessandro Aldini, Gilles Barthe e Roberto Gorrieri. Berlin: Springer Berlin Heidelberg, 2009. p. 145–165. DOI: 10.1007/978-3-642-03829-7_3. Citado na p. 18.

ASF. **Apache CloudStack**. [S.l.]: The Apache Software Foundation, 2024. Disponível em: <https://cloudstack.apache.org/>. Acesso em: 10 set. 2024. Citado nas pp. 15, 23.

ASF. **Apache Syncope**. [S.l.]: The Apache Software Foundation, 2024. Disponível em: <https://syncope.apache.org/>. Acesso em: 20 out. 2024. Citado na p. 22.

ASF. **Managing Accounts, Users and Domains**. [S.l.]: The Apache Software Foundation, 2024. Disponível em: <https://docs.cloudstack.apache.org/en/4.19.1.1/adminguide/accounts.html>. Acesso em: 20 set. 2024. Citado nas pp. 23–25.

AUTHELIA. **Authelia - The Single Sign-On Multi-Factor portal for web apps**. [S.l.]: Authelia, 2024. Disponível em: <https://www.authelia.com/>. Acesso em: 20 out. 2024. Citado na p. 22.

BERTINO, Elisa; TAKAHASHI, Kenji. **Identity Management**: Concepts, Technologies, and Systems. Norwood, MA: Artech House, 2011. Citado nas pp. 17, 19.

BHARGAV-SPANTZEL, Abhilasha et al. User centricity: A taxonomy and open issues. **Journal of Computer Security**, IOS Press, Amsterdam, v. 15, n. 5, p. 493–527, out. 2007. DOI: 10.3233/jcs-2007-15502. Citado na p. 17.

CCITT. Security architecture for Open Systems Interconnection for CCITT applications. **Recommendation X.800**, International Telecommunication Union, Geneva, 1991. Disponível em: <https://handle.itu.int/11.1002/1000/3102>. Citado na p. 18.

CHA, ByungRae; SEO, JaeHyun; KIM, JongWon. Design of Attribute-Based Access Control in Cloud Computing Environment. In: KIM, Kuinam J.; AHN, Seong Jin (ed.). **Proceedings of the International Conference on IT Convergence and Security 2011**. Dordrecht: Springer Science+Business Media, 2012. p. 41–50. DOI: 10.1007/978-94-007-2911-7_4. Citado na p. 15.

CHADWICK, David W. Federated Identity Management. In: **Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures**. Edição: Alessandro Aldini, Gilles Barthe e Roberto Gorrieri. Berlin: Springer Berlin Heidelberg, 2009. p. 96–120. DOI: 10.1007/978-3-642-03829-7_3. Citado nas pp. 17, 19.

CHADWICK, David W.; INMAN, George. Attribute Aggregation in Federated Identity Management. **Computer**, IEEE Computer Society, v. 42, n. 5, p. 33–40, 2009. DOI: 10.1109/MC.2009.143. Citado nas pp. 19, 50.

COYNE, Ed; WEIL, Timothy R. ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. **IT Professional**, IEEE Computer Society, Los Alamitos, v. 15, n. 03, p. 14–16, maio 2013. DOI: 10.1109/MITP.2013.37. Citado nas pp. 18, 19.

FERRAIOLO, David F. et al. Proposed NIST standard for role-based access control. **ACM Transactions on Information and System Security**, Association for Computing Machinery, New York, v. 4, n. 3, p. 224–274, 2001. Citado nas pp. 18, 19.

HU, Vincent C. et al. Attribute-Based Access Control. **Computer**, IEEE Computer Society, v. 48, n. 2, p. 85–88, 2015. DOI: 10.1109/MC.2015.33. Citado na p. 15.

INDU, I.; ANAND, P. M. Rubesh; BHASKAR, Vidhyacharan. Identity and access management in cloud environment: Mechanisms and challenges. **Engineering Science and Technology, an International Journal**, v. 21, n. 4, p. 574–588, 2018. DOI: 10.1016/j.jestch.2018.05.010. Citado nas pp. 15, 18.

ITU-T. Authentication and authorization requirements for NGN release 1. **Recommendation ITU-T Y.2702**, International Telecommunication Union, Geneva, 2009. Disponível em: <https://handle.itu.int/11.1002/1000/9359>. Citado na p. 18.

ITU-T. Information technology - Open Systems Interconnection - Security frameworks for open systems: Authentication framework. **ITU-T Recommendation X.811**, International Telecommunication Union, 1996. Disponível em: <https://handle.itu.int/11.1002/1000/3107>. Citado nas pp. 17, 18.

ITU-T. NGN identity management framework. **Recommendation ITU-T Y.2720**, International Telecommunication Union, Geneva, 2009. Disponível em: <https://handle.itu.int/11.1002/1000/9574>. Citado nas pp. 15, 17, 19.

ITU-T. Security Assertion Markup Language (SAML 2.0). **ITU-T Recommendation X.1141**, International Telecommunication Union, Geneva, 2007. Disponível em: <https://handle.itu.int/11.1002/1000/3102>. Citado na p. 20.

JØSANG, Audun. A Consistent Definition of Authorization. In: 13. INTERNATIONAL Workshop on Security and Trust Management. Oslo: [s.n.], set. 2017. p. 134–144. DOI: 10.1007/978-3-319-68063-7_9. Citado na p. 18.

KUHN, D. Richard; COYNE, Edward J.; WEIL, Timothy R. Adding Attributes to Role-Based Access Control. **Computer**, IEEE Computer Society, v. 43, n. 6, p. 79–81, 2010. DOI: 10.1109/MC.2010.155. Citado nas pp. 30, 50.

LAMPSON, Butler et al. Authentication in Distributed Systems: Theory and Practice. **ACM Transactions on Computer Systems**, Association for Computing Machinery, v. 10, p. 265–310, nov. 1992. DOI: 10.1145/138873.138874. Citado na p. 18.

MARTINS, Pedro Henrique Pereira. **Proposta de um framework autônomo para monitoramento de ambientes de computação em nuvem**. TCC (Bacharelado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2016. Disponível em: <https://repositorio.ufsc.br/handle/123456789/132453>. Citado na p. 23.

MELL, Peter; GRANCE, Timothy. The NIST Definition of Cloud Computing. **Special Publication 800-145**, National Institute of Standards e Technology, Gaithersburg, 2011. DOI: 10.6028/NIST.SP.800-145. Citado nas pp. 22, 23.

MIERS, Charles Christian et al. Mecanismos de autenticação e autorização para nuvens computacionais: definição, classificação e análise de soluções. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, XXXV. 2017, Belém. *Minicursos*. Porto Alegre: Sociedade Brasileira de Computação, 2017. p. 203–246. Citado nas pp. 26, 30, 31.

OPENSTACK. **Introduction to Keystone Federation**. [S.l.]: OpenInfra Foundation, 2024. Disponível em: <https://docs.openstack.org/keystone/2024.1/admin/federation/introduction.html>. Acesso em: 10 set. 2024. Citado nas pp. 27–29.

OPENSTACK. **Keystone Architecture**. [S.l.]: OpenInfra Foundation, 2024. Disponível em: <https://docs.openstack.org/keystone/2024.1/getting-started/architecture.html>. Acesso em: 20 set. 2024. Citado na p. 27.

OPENSTACK. **OpenStack**. [S.l.]: OpenInfra Foundation, 2024. Disponível em: <https://www.openstack.org/>. Acesso em: 10 set. 2024. Citado na p. 26.

OPENSTACK. Policies. In: OPENSTACK Security Guide. [S.l.]: OpenInfra Foundation, 2024. Disponível em: <https://docs.openstack.org/security-guide/identity/policies.html>. Acesso em: 20 set. 2024. Citado na p. 27.

SAKIMURA, Nat et al. **OpenID Connect Core 1.0 incorporating errata set 2**. [S.l.]: OpenID Foundation, 2023. Disponível em: <https://openid.net/specs/openid-connect-core-1_0.html>. Acesso em: 20 set. 2024. Citado na p. 21.

THORGERSEN, Stian; SILVA, Pedro Igor. **Keycloak - Identity and Access Management for Modern Applications**: Harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications. Birmingham: Packt, 2021. Citado na p. 22.

WEINGÄRTNER, Rafael. **Controle de disseminação de dados sensíveis em ambientes federados**. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2014. Disponível em: <https://repositorio.ufsc.br/handle/123456789/132453>. Citado nas pp. 18, 31, 32, 51.

WEINGÄRTNER, Rafael. **Keystone identity mapping to support project definition as a JSON**. [S.l.]: OpenDev, 2020. Disponível em: <https://review.opendev.org/c/openstack/keystone/+/742235>. Acesso em: 20 set. 2024. Citado na p. 27.

WERNER, Jorge. **Uma abordagem de privacidade no gerenciamento de identidade na nuvem**. Tese (Doutorado em Ciência da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2017. Disponível em: <https://repositorio.ufsc.br/handle/123456789/188839>. Citado nas pp. 32, 33.

WILSON, Yvonne; HINGNIKAR, Abhishek. **Solving Identity Management in Modern Applications**: Demistifying OAuth 2.0, OpenID Connect, and SAML 2.0. New York: Apress, 2019. DOI: 10.1007/978-1-4842-5095-2. Citado nas pp. 15, 17, 19–21.

ZITADEL. **ZITADEL - Identity infrastructure, simplified for you**. San Francisco: ZITADEL, 2024. Disponível em: <https://zitadel.com/>. Acesso em: 20 out. 2024. Citado na p. 22.

Apêndices

APÊNDICE A – CÓDIGO-FONTE

Estas são as alterações descritas no Capítulo 5. Foram realizadas no código-fonte da versão **4.20.0.0**[1], presente no repositório oficial do projeto Apache CloudStack[2]. Os arquivos modificados estão apresentados de acordo com o resultado do comando `git diff`, enquanto os arquivos criados estão dispostos em sua totalidade.

A.1 API/SRC/MAIN/JAVA/COM/CLOUD/EVENT/ EVENTTYPES.JAVA

```
diff --git a/api/src/main/java/com/cloud/event/EventTypes.java b/api/src/main/java/
   ↪com/cloud/event/EventTypes.java
index 5e5309965c..21c8d92dd3 100644
--- a/api/src/main/java/com/cloud/event/EventTypes.java
+++ b/api/src/main/java/com/cloud/event/EventTypes.java
@@ -27,6 +27,7 @@ import org.apache.cloudstack.api.response.ClusterResponse;
 import org.apache.cloudstack.api.response.HostResponse;
 import org.apache.cloudstack.api.response.PodResponse;
 import org.apache.cloudstack.api.response.ZoneResponse;
+import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
 import org.apache.cloudstack.config.Configuration;
 import org.apache.cloudstack.datacenter.DataCenterIpv4GuestSubnet;
 import org.apache.cloudstack.ha.HAConfig;
@@ -784,6 +785,11 @@ public class EventTypes {
     public static final String EVENT_SHAREDFS_EXPUNGE = "SHAREDFS.EXPUNGE";
     public static final String EVENT_SHAREDFS_RECOVER = "SHAREDFS.RECOVER";

+    // IdP Sync Policy
+    public static final String EVENT_IDPSYNCPOLICY_CREATE = "IDPSYNCPOLICY.CREATE";
+    public static final String EVENT_IDPSYNCPOLICY_REMOVE = "IDPSYNCPOLICY.REMOVE";
+    public static final String EVENT_IDPSYNCPOLICY_UPDATE = "IDPSYNCPOLICY.UPDATE";
+
     static {

         // TODO: need a way to force author adding event types to declare the
   ↪entity details as well, with out braking
@@ -1272,6 +1278,11 @@ public class EventTypes {
```

[1] *commit* `2fe3fcef7c77cf9a1b629c9df3afc0cdb88ad4f6` da *branch* `main` (marcado com a *tag* `4.20.0.0`)

[2] https://github.com/apache/cloudstack

```
         entityEventDetails.put(EVENT_SHAREDFS_DESTROY, SharedFS.class);
         entityEventDetails.put(EVENT_SHAREDFS_EXPUNGE, SharedFS.class);
         entityEventDetails.put(EVENT_SHAREDFS_RECOVER, SharedFS.class);
+
+        // Idp Sync Policy
+        entityEventDetails.put(EVENT_IDPSYNCPOLICY_CREATE, IdpSyncPolicy.class);
+        entityEventDetails.put(EVENT_IDPSYNCPOLICY_REMOVE, IdpSyncPolicy.class);
+        entityEventDetails.put(EVENT_IDPSYNCPOLICY_UPDATE, IdpSyncPolicy.class);
     }

     public static boolean isNetworkEvent(String eventType) {
```

A.2 API/SRC/MAIN/JAVA/COM/CLOUD/USER/ ACCOUNTSERVICE.JAVA

```
diff --git a/api/src/main/java/com/cloud/user/AccountService.java b/api/src/main/
   ↪java/com/cloud/user/AccountService.java
index 60db7abb73..d238006b0e 100644
--- a/api/src/main/java/com/cloud/user/AccountService.java
+++ b/api/src/main/java/com/cloud/user/AccountService.java
@@ -45,7 +45,8 @@ public interface AccountService {
     UserAccount createUserAccount(CreateAccountCmd accountCmd);

     UserAccount createUserAccount(String userName, String password, String
   ↪firstName, String lastName, String email, String timezone, String accountName,
   ↪Account.Type accountType,
-                                  Long roleId, Long domainId, String networkDomain,
   ↪ Map<String, String> details, String accountUUID, String userUUID, User.Source
   ↪source);
+                                  Long roleId, Long domainId, String networkDomain,
   ↪ Map<String, String> details, String accountUUID, String userUUID, String
   ↪externalEntity,
+                                  User.Source source);

     /**
      * Locks a user by userId. A locked user cannot access the API, but will still
   ↪have running VMs/IP addresses
@@ -60,7 +61,7 @@ public interface AccountService {
     User createUser(String userName, String password, String firstName, String
   ↪lastName, String email, String timeZone, String accountName, Long domainId,
```

```
    ↪String userUUID);

     User createUser(String userName, String password, String firstName, String
    ↪lastName, String email, String timeZone, String accountName, Long domainId,
    ↪String userUUID,
-                    User.Source source);
+                    String externalEntity, User.Source source);

     boolean isAdmin(Long accountId);
```

A.3 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/APICOMMANDRESOURCETYPE.JAVA

```
diff --git a/api/src/main/java/org/apache/cloudstack/api/ApiCommandResourceType.java
    ↪ b/api/src/main/java/org/apache/cloudstack/api/ApiCommandResourceType.java
index f2f52cec96..0141a498ec 100644
--- a/api/src/main/java/org/apache/cloudstack/api/ApiCommandResourceType.java
+++ b/api/src/main/java/org/apache/cloudstack/api/ApiCommandResourceType.java
@@ -21,6 +21,7 @@ import java.util.HashMap;
 import java.util.List;
 import java.util.Map;

+import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
 import org.apache.cloudstack.region.PortableIp;
 import org.apache.commons.collections.CollectionUtils;
 import org.apache.commons.lang3.EnumUtils;
@@ -86,7 +87,8 @@ public enum ApiCommandResourceType {
     QuotaTariff(org.apache.cloudstack.quota.QuotaTariff.class),
     KubernetesCluster(null),
     KubernetesSupportedVersion(null),
-    SharedFS(org.apache.cloudstack.storage.sharedfs.SharedFS.class);
+    SharedFS(org.apache.cloudstack.storage.sharedfs.SharedFS.class),
+    IdpSyncPolicy(org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy.class);

     private final Class<?> clazz;
```

A.4 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/APICONSTANTS.JAVA

```
diff --git a/api/src/main/java/org/apache/cloudstack/api/ApiConstants.java b/api/src
```

```
   ↪/main/java/org/apache/cloudstack/api/ApiConstants.java
index bb16b0ff90..104b9acfa1 100644
--- a/api/src/main/java/org/apache/cloudstack/api/ApiConstants.java
+++ b/api/src/main/java/org/apache/cloudstack/api/ApiConstants.java
@@ -318,6 +318,7 @@ public class ApiConstants {
     public static final String LBID = "lbruleid";
     public static final String LB_PROVIDER = "lbprovider";
     public static final String MAC_ADDRESS = "macaddress";
+    public static final String MAPPING = "mapping";
     public static final String MAX = "max";
     public static final String MAX_SNAPS = "maxsnaps";
     public static final String MAX_CPU_NUMBER = "maxcpunumber";
@@ -498,8 +499,8 @@ public class ApiConstants {
     public static final String URL = "url";
     public static final String USAGE_INTERFACE = "usageinterface";
     public static final String USED_SUBNETS = "usedsubnets";
+    public static final String USERACCOUNT_OPERATION = "useraccount_operation";
     public static final String USER_DATA = "userdata";
-
     public static final String USER_DATA_NAME = "userdataname";
     public static final String USER_DATA_ID = "userdataid";
     public static final String USER_DATA_POLICY = "userdatapolicy";
@@ -549,6 +550,7 @@ public class ApiConstants {
     public static final String VOLUMES = "volumes";
     public static final String VOLUME_CHECK_RESULT = "volumecheckresult";
     public static final String VOLUME_REPAIR_RESULT = "volumerepairresult";
+    public static final String WARNING = "warning";

     public static final String ZONE = "zone";
     public static final String ZONE_ID = "zoneid";
@@ -926,6 +928,7 @@ public class ApiConstants {
     public static final String VMPROFILE_ID = "vmprofileid";
     public static final String VMGROUP_ID = "vmgroupid";
     public static final String CS_URL = "csurl";
+    public static final String IDP = "idp";
     public static final String IDP_ID = "idpid";
     public static final String SCALEUP_POLICY_IDS = "scaleuppolicyids";
     public static final String SCALEDOWN_POLICY_IDS = "scaledownpolicyids";
@@ -1179,6 +1182,15 @@ public class ApiConstants {
```

```
            "numeric value will be applied; if the result is neither a boolean nor
    ↪a numeric value, the tariff will not be applied. If the rule is not informed,
    ↪the tariff " +
            "value will be applied.";

+    public static final String PARAMETER_DESCRIPTION_IDPSYNCPOLICY_MAPPING = "
    ↪JavaScript script that returns an object with specific attributes. Necessary
    ↪attributes vary according to " +
+           USERACCOUNT_OPERATION + ".";
+
+    public static final String PARAMETER_DESCRIPTION_USERACCOUNT_OPERATION = "Which
    ↪ operation to execute after resolving the script. " +
+           "Value must be one of: NONE (do nothing if rule result can't find a
    ↪user to login, do nothing if found user has different attributes to those
    ↪received from IdP); " +
+           "CREATE (create user if not found during login, do nothing if found
    ↪user has different attributes to those received from IdP); " +
+           "UPDATE (do nothing if rule result can't find a user to login, update
    ↪user if it has different attributes to those received from IdP); " +
+           "CREATEANDUPDATE (create user if not found during login, update user if
    ↪ it has different attributes to those received from IdP).";
+
     /**
      * This enum specifies IO Drivers, each option controls specific policies on I/
    ↪O.
      * Qemu guests support "threads" and "native" options Since 0.8.8 ; "io_uring"
    ↪is supported Since 6.3.0 (QEMU 5.0).
```

A.5 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/ACCOUNT/ UPDATEACCOUNTCMD.JAVA

```
diff --git a/api/src/main/java/org/apache/cloudstack/api/command/admin/account/
    ↪UpdateAccountCmd.java b/api/src/main/java/org/apache/cloudstack/api/command/
    ↪admin/account/UpdateAccountCmd.java
index 91cbb90e4d..dd1af5f89c 100644
--- a/api/src/main/java/org/apache/cloudstack/api/command/admin/account/
    ↪UpdateAccountCmd.java
+++ b/api/src/main/java/org/apache/cloudstack/api/command/admin/account/
    ↪UpdateAccountCmd.java
```

```
@@ -139,6 +139,26 @@ public class UpdateAccountCmd extends BaseCmd {
         }
     }

+    public void setId(Long accountId) {
+        this.id = accountId;
+    }
+
+    public void setNewName(String newName) {
+        this.newName = newName;
+    }
+
+    public void setNetworkDomain(String networkDomain) {
+        this.networkDomain = networkDomain;
+    }
+
+    public void setDetails(Map details) {
+        this.details = details;
+    }
+
+    public void setRoleId(Long roleId) {
+        this.roleId = roleId;
+    }
+
     @Override
     public Long getApiResourceId() {
         return id;
```

A.6 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/ IDPSYNCPOLICY/ CREATEIDPSYNCPOLICYCMD.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
```

```
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.api.command.admin.idpsyncpolicy;

import com.cloud.exception.InvalidParameterValueException;
import com.cloud.user.Account;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.api.APICommand;
import org.apache.cloudstack.api.ApiCommandResourceType;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.ApiErrorCode;
import org.apache.cloudstack.api.BaseCmd;
import org.apache.cloudstack.api.Parameter;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.response.IdpSyncPolicyResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;

import javax.inject.Inject;

@APICommand(name = "createIdpSyncPolicy", responseObject = IdpSyncPolicyResponse.
    ↪class, since = "4.20", entityType = IdpSyncPolicyResponse.class, authorized = {
    ↪RoleType.Admin},
        description = "Creates an IdP Sync Policy to receive attributes from an
    ↪Identity Provider.")
public class CreateIdpSyncPolicyCmd extends BaseCmd {

    @Parameter(name = ApiConstants.IDP_ID, type = CommandType.STRING, length = 65535,
    ↪ required = true, description = "Entity ID of IdP which will send attributes.")
    private String idpId;

    @Parameter(name = ApiConstants.DESCRIPTION, type = CommandType.STRING, length =
```

```
↪65535, description = "Optional description for operator use.")
private String description;

@Parameter(name = ApiConstants.MAPPING, type = CommandType.STRING, required =
↪true, length = 65535, description = ApiConstants.
↪PARAMETER_DESCRIPTION_IDPSYNCPOLICY_MAPPING)
private String mapping;

@Parameter(name = ApiConstants.USERACCOUNT_OPERATION, type = CommandType.STRING,
↪ description = ApiConstants.PARAMETER_DESCRIPTION_USERACCOUNT_OPERATION)
private String userAccountOperation;

@Inject
private IdpSyncPolicyManager idpSyncPolicyManager;

public String getIdpId() {
    return idpId;
}

public String getDescription() {
    return description;
}

public String getMapping() {
    return mapping;
}

public IdpSyncPolicy.Operation getUserAccountOperation() throws
↪InvalidParameterValueException {
    if (userAccountOperation == null || userAccountOperation.isBlank()) {
        return IdpSyncPolicy.Operation.NONE;
    }
    try {
        return IdpSyncPolicy.Operation.valueOf(userAccountOperation.toUpperCase
↪());
    } catch (IllegalArgumentException e) {
        logger.error("Parameter [{}] does not accept [{}].", ApiConstants.
↪USERACCOUNT_OPERATION, userAccountOperation);
        throw new InvalidParameterValueException(String.format("Operation [%s]
```

```
↪does not exist.", userAccountOperation));
        }
    }

    @Override
    public void execute() throws ServerApiException, InvalidParameterValueException
↪{
        IdpSyncPolicy idpSyncPolicy = idpSyncPolicyManager.createIdpSyncPolicy(this)
↪;
        if (idpSyncPolicy == null) {
            throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Failed to
↪create an IdP Sync Policy.");
        }

        IdpSyncPolicyResponse response = new IdpSyncPolicyResponse(idpSyncPolicy);
        response.setResponseName(getCommandName());
        this.setResponseObject(response);
    }

    public void setUserAccountOperation(String userAccountOperation) {
        this.userAccountOperation = userAccountOperation;
    }

    public void setIdpId(String idpId) {
        this.idpId = idpId;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setMapping(String mapping) {
        this.mapping = mapping;
    }

    @Override
    public long getEntityOwnerId() {
        return Account.ACCOUNT_ID_SYSTEM;
    }
```

```
    @Override
    public ApiCommandResourceType getApiResourceType() {
        return ApiCommandResourceType.IdpSyncPolicy;
    }
}
```

A.7 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/IDPSYNCPOLICY/LISTIDPSYNCPOLICIESCMD.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.api.command.admin.idpsyncpolicy;

import com.cloud.user.Account;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.api.APICommand;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.BaseListCmd;
import org.apache.cloudstack.api.Parameter;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.response.IdpSyncPolicyResponse;
import org.apache.cloudstack.api.response.ListResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
```

```
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;

import javax.inject.Inject;
import java.util.ArrayList;
import java.util.List;

@APICommand(name = "listIdpSyncPolicies", responseObject = IdpSyncPolicyResponse.
    ↪class, since = "4.20", entityType = IdpSyncPolicyResponse.class, authorized = {
    ↪RoleType.Admin},
        description = "Lists IdP Sync Policies.")
public class ListIdpSyncPoliciesCmd extends BaseListCmd {

    @Parameter(name = ApiConstants.ID, type = CommandType.UUID, entityType =
    ↪IdpSyncPolicyResponse.class, description = "ID of the IdP Sync Policy.")
    private Long id;

    @Parameter(name = ApiConstants.IDP_ID, type = CommandType.STRING, description =
    ↪"IdP related to the Sync Policies.")
    private String idpId;

    @Parameter(name = ApiConstants.SHOW_REMOVED, type = CommandType.BOOLEAN,
    ↪description = "Whether to show deleted policies.")
    private boolean showRemoved = false;

    @Inject
    private IdpSyncPolicyManager idpSyncPolicyManager;

    public Long getId() {
        return id;
    }

    public String getIdpId() {
        return idpId;
    }

    public boolean showRemoved() {
        return showRemoved;
    }
```

```
    @Override
    public void execute() throws ServerApiException {
        List<? extends IdpSyncPolicy> policies = idpSyncPolicyManager.
    ↪listIdpSyncPolicies(this);

        List<IdpSyncPolicyResponse> responses = new ArrayList<>();
        for (IdpSyncPolicy policy : policies) {
            responses.add(new IdpSyncPolicyResponse(policy));
        }

        ListResponse<IdpSyncPolicyResponse> listResponse = new ListResponse<>();
        listResponse.setResponses(responses);
        listResponse.setResponseName(getCommandName());
        this.setResponseObject(listResponse);
    }

    @Override
    public long getEntityOwnerId() {
        return Account.ACCOUNT_ID_SYSTEM;
    }
}
```

A.8 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/IDPSYNCPOLICY/REMOVEIDPSYNCPOLICYCMD.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
```

```
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.api.command.admin.idpsyncpolicy;

import com.cloud.user.Account;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.api.APICommand;
import org.apache.cloudstack.api.ApiCommandResourceType;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.BaseCmd;
import org.apache.cloudstack.api.Parameter;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.response.IdpSyncPolicyResponse;
import org.apache.cloudstack.api.response.SuccessResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;

import javax.inject.Inject;

@APICommand(name = "removeIdpSyncPolicy", responseObject = IdpSyncPolicyResponse.
    ↪class, since = "4.20", entityType = IdpSyncPolicyResponse.class, authorized = {
    ↪RoleType.Admin},
        description = "Removes an IdP Sync Policy.")
public class RemoveIdpSyncPolicyCmd extends BaseCmd {

    @Parameter(name = ApiConstants.ID, type = CommandType.UUID, entityType =
    ↪IdpSyncPolicyResponse.class, required = true, description = "ID of the IdP Sync
    ↪ Policy to be removed.")
    private Long id;

    @Parameter(name = ApiConstants.REMOVAL_REASON, type = CommandType.STRING,
    ↪required = true, description = "Reason for removing.")
    private String removalReason;

    @Inject
    private IdpSyncPolicyManager idpSyncPolicyManager;

    public Long getId() {
```

```
        return id;
    }

    public String getRemovalReason() {
        return removalReason;
    }

    @Override
    public void execute() throws ServerApiException {
        IdpSyncPolicy idpSyncPolicy = idpSyncPolicyManager.removeIdpSyncPolicy(this)
    ↪;

        SuccessResponse response = new SuccessResponse();
        response.setResponseName(getCommandName());
        this.setResponseObject(response);
    }

    @Override
    public long getEntityOwnerId() {
        return Account.ACCOUNT_ID_SYSTEM;
    }

    @Override
    public Long getApiResourceId() {
        return id;
    }

    @Override
    public ApiCommandResourceType getApiResourceType() {
        return ApiCommandResourceType.IdpSyncPolicy;
    }
}
```

A.9 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/IDPSYNCPOLICY/UPDATEIDPSYNCPOLICYCMD.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
```

```
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.api.command.admin.idpsyncpolicy;

import com.cloud.exception.InvalidParameterValueException;
import com.cloud.user.Account;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.api.APICommand;
import org.apache.cloudstack.api.ApiCommandResourceType;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.ApiErrorCode;
import org.apache.cloudstack.api.BaseCmd;
import org.apache.cloudstack.api.Parameter;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.response.IdpSyncPolicyResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;

import javax.inject.Inject;

@APICommand(name = "updateIdpSyncPolicy", responseObject = IdpSyncPolicyResponse.
    ↪class, since = "4.20", entityType = IdpSyncPolicyResponse.class, authorized = {
    ↪RoleType.Admin},
        description = "Updates an IdP Sync Policy.")
public class UpdateIdpSyncPolicyCmd extends BaseCmd {

    @Parameter(name = ApiConstants.ID, type = CommandType.UUID, entityType =
```

```
↪IdpSyncPolicyResponse.class, required = true, description = "ID of the IdP Sync
↪ Policy to be updated.")
private Long id;

@Parameter(name = ApiConstants.DESCRIPTION, type = CommandType.STRING, length =
↪65535, description = "Optional description for operator use.")
private String description;

@Parameter(name = ApiConstants.MAPPING, type = CommandType.STRING, length =
↪65535, description = ApiConstants.PARAMETER_DESCRIPTION_IDPSYNCPOLICY_MAPPING)
private String mapping;

@Inject
private IdpSyncPolicyManager idpSyncPolicyManager;

public Long getId() {
    return id;
}

public String getDescription() {
    return description;
}

public String getMapping() {
    return mapping;
}


@Override
public void execute() throws ServerApiException, InvalidParameterValueException
↪{
    IdpSyncPolicy idpSyncPolicy = idpSyncPolicyManager.updateIdpSyncPolicy(this)
↪;
    if (idpSyncPolicy == null) {
        throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Error
↪updating IdP Sync Policy.");
    }

    IdpSyncPolicyResponse response = new IdpSyncPolicyResponse(idpSyncPolicy);
```

```
        response.setResponseName(getCommandName());
        this.setResponseObject(response);
    }

    @Override
    public long getEntityOwnerId() {
        return Account.ACCOUNT_ID_SYSTEM;
    }

    @Override
    public Long getApiResourceId() {
        return id;
    }

    @Override
    public ApiCommandResourceType getApiResourceType() {
        return ApiCommandResourceType.IdpSyncPolicy;
    }
}
```

A.10 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/IDPSYNCPOLICY/VALIDATEIDPSYNCPOLICYMAPPINGCMD.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
```

```
package org.apache.cloudstack.api.command.admin.idpsyncpolicy;

import com.cloud.exception.InvalidParameterValueException;
import com.cloud.user.Account;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.api.APICommand;
import org.apache.cloudstack.api.ApiCommandResourceType;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.ApiErrorCode;
import org.apache.cloudstack.api.BaseCmd;
import org.apache.cloudstack.api.Parameter;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.response.IdpSyncPolicyMappingResponse;
import org.apache.cloudstack.api.response.IdpSyncPolicyResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;

import javax.inject.Inject;

@APICommand(name = "validateIdpSyncPolicyMapping", responseObject =
    ↪IdpSyncPolicyMappingResponse.class, since = "4.20", entityType =
    ↪IdpSyncPolicyResponse.class,
        authorized = {RoleType.Admin}, description = "Validates provided IdP Sync
    ↪Policy and returns the mapping script result.")
public class ValidateIdpSyncPolicyMappingCmd extends BaseCmd {

    @Parameter(name = ApiConstants.IDP, type = CommandType.STRING, required = true,
    ↪description = "JSON object containing simulating the attributes provided by the
    ↪ IdP.")
    private String idpJson;

    @Parameter(name = ApiConstants.MAPPING, type = CommandType.STRING, required =
    ↪true, description = ApiConstants.PARAMETER_DESCRIPTION_IDPSYNCPOLICY_MAPPING)
    private String mapping;

    @Parameter(name = ApiConstants.USERACCOUNT_OPERATION, type = CommandType.STRING,
    ↪ description = ApiConstants.PARAMETER_DESCRIPTION_USERACCOUNT_OPERATION)
    private String userAccountOperation;
```

```
@Inject
private IdpSyncPolicyManager idpSyncPolicyManager;

public ValidateIdpSyncPolicyMappingCmd() {
}

public String getIdpJson() {
    return idpJson;
}

public String getMapping() {
    return mapping;
}

public IdpSyncPolicy.Operation getUserAccountOperation() throws
↪InvalidParameterValueException {
    if (userAccountOperation == null || userAccountOperation.isBlank()) {
        return IdpSyncPolicy.Operation.NONE;
    }
    try {
        return IdpSyncPolicy.Operation.valueOf(userAccountOperation.toUpperCase
↪());
    } catch (IllegalArgumentException e) {
        logger.error("Parameter [{}] does not accept [{}].", ApiConstants.
↪USERACCOUNT_OPERATION, userAccountOperation);
        throw new InvalidParameterValueException(String.format("Operation [%s]
↪does not exist.", userAccountOperation));
    }
}

@Override
public void execute() throws ServerApiException, InvalidParameterValueException
↪{
    IdpSyncPolicyMappingResponse response = idpSyncPolicyManager.
↪validateIdpSyncPolicyMapping(this);
    if (response == null) {
        throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Error
↪validating IdP Sync Policy mapping.");
```

```
        }
        response.setResponseName(getCommandName());
        this.setResponseObject(response);
    }

    @Override
    public long getEntityOwnerId() {
        return Account.ACCOUNT_ID_SYSTEM;
    }

    @Override
    public ApiCommandResourceType getApiResourceType() {
        return ApiCommandResourceType.IdpSyncPolicy;
    }
}
```

A.11 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/ADMIN/USER/ UPDATEUSERCMD.JAVA

```
diff --git a/api/src/main/java/org/apache/cloudstack/api/command/admin/user/
   ↪UpdateUserCmd.java b/api/src/main/java/org/apache/cloudstack/api/command/admin/
   ↪user/UpdateUserCmd.java
index c9e1e93415..8e3c25b8e7 100644
--- a/api/src/main/java/org/apache/cloudstack/api/command/admin/user/UpdateUserCmd.
   ↪java
+++ b/api/src/main/java/org/apache/cloudstack/api/command/admin/user/UpdateUserCmd.
   ↪java
@@ -176,6 +176,16 @@ public class UpdateUserCmd extends BaseCmd {
         this.email = email;
     }

+    public void setUsername(String username) {
+        this.username = username;
+    }
+
+    public void setTimezone(String timezone) {
+        this.timezone = timezone;
+    }
+
```

```
+
+
     @Override
     public Long getApiResourceId() {
         return id;
```

A.12 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/RESPONSE/IDPSYNCPOLICYMAPPINGRESPONSE.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
package org.apache.cloudstack.api.response;

import com.cloud.serializer.Param;
import com.google.gson.annotations.SerializedName;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.AttributeMap;

public class IdpSyncPolicyMappingResponse extends SuccessResponse {
    @SerializedName("mappedattributes")
    @Param(description = "Identified and mapped attributes.")
    private AttributeMap mappedAttributes;

    @SerializedName("userresponse")
    @Param(description = "User found during login simulation.")
    private UserResponse userResponse;
```

```
    public AttributeMap getMappedAttributes() {
        return mappedAttributes;
    }

    public void setMappedAttributes(AttributeMap mappedAttributes) {
        this.mappedAttributes = mappedAttributes;
    }

    public UserResponse getUserResponse() {
        return userResponse;
    }

    public void setUserResponse(UserResponse userResponse) {
        this.userResponse = userResponse;
    }

    public IdpSyncPolicyMappingResponse() {
    }

    public IdpSyncPolicyMappingResponse(String responseName) {
        super.setResponseName(responseName);
    }
}
```

A.13 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/RESPONSE/IDPSYNCPOLICYRESPONSE.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
```

```
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.api.response;


import com.cloud.serializer.Param;
import com.google.gson.annotations.SerializedName;
import org.apache.cloudstack.api.ApiConstants;
import org.apache.cloudstack.api.BaseResponse;
import org.apache.cloudstack.api.EntityReference;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicy;

import java.util.Date;


@EntityReference(value = IdpSyncPolicy.class)
public class IdpSyncPolicyResponse extends BaseResponse {
    @SerializedName(ApiConstants.ID)
    @Param(description = "ID of the IdP Sync Policy.")
    private String id;

    @SerializedName(ApiConstants.DESCRIPTION)
    @Param(description = "Description of the IdP Sync Policy.")
    private String description;

    @SerializedName(ApiConstants.IDP_ID)
    @Param(description = "IdP ID of the IdP Sync Policy.")
    private String idpId;

    @SerializedName(ApiConstants.USERACCOUNT_OPERATION)
    @Param(description = "UserAccount Operation of the IdP Sync Policy.")
    private String userAccountOperation;

    @SerializedName(ApiConstants.MAPPING)
    @Param(description = "Mapping script of the IdP Sync Policy.")
    private String mapping;
```

```
    @SerializedName(ApiConstants.CREATED)
    @Param(description = "Date the IdP Sync Policy was created.")
    private Date created;

    @SerializedName(ApiConstants.LAST_UPDATED)
    @Param(description = "Date the IdP Sync Policy was last updated.")
    private Date updated;

    @SerializedName("updatecount")
    @Param(description = "Number of times the IdP Sync policy was updated.")
    private Long updateCount;

    @SerializedName(ApiConstants.REMOVED)
    @Param(description = "Date the IdP Sync Policy was deleted.")
    private Date removed;

    @SerializedName(ApiConstants.REMOVAL_REASON)
    @Param(description = "Reason for IdP Sync Policy being deleted.")
    private String removalReason;

    public IdpSyncPolicyResponse(IdpSyncPolicy idpSyncPolicy) {
        this.id = idpSyncPolicy.getUuid();
        this.description = idpSyncPolicy.getDescription();
        this.idpId = idpSyncPolicy.getIdpId();
        this.userAccountOperation = idpSyncPolicy.getOperation().toString();
        this.mapping = idpSyncPolicy.getMapping();
        this.created = idpSyncPolicy.getCreated();
        this.updated = idpSyncPolicy.getUpdated();
        if (idpSyncPolicy.getUpdated() != null) {
            this.updateCount = idpSyncPolicy.getUpdateCount();
        }
        this.removed = idpSyncPolicy.getRemoved();
        if (idpSyncPolicy.getRemoved() != null) {
            this.removalReason = idpSyncPolicy.getRemovalReason();
        }
        this.setObjectName("idpsyncpolicy");
    }
}
```

A.14 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/IDPSYNCPOLICY.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
package org.apache.cloudstack.auth.idpsyncpolicy;

import org.apache.cloudstack.api.Identity;
import org.apache.cloudstack.api.InternalIdentity;

import java.util.Date;

public interface IdpSyncPolicy extends Identity, InternalIdentity {

    /**
     * <b>NONE</b>: Just tries to log in, will not care about discrepancies.<br>
     * <b>CREATE</b>: Tries to log in. If user and/or account not found, will try to
    ↪ create them.<br>
     * <b>UPDATE</b>: Tries to log in. If user is found, but has differing
    ↪attributes, will try to update them.<br>
     * <b>CREATEANDUPDATE</b>: Both CREATE and UPDATE.<br>
     */
    enum Operation {
        NONE, CREATE, UPDATE, CREATEANDUPDATE
    }
```

```
    String getIdpId();

    Operation getOperation();

    String getMapping();
    void setMapping(String mapping);

    String getDescription();
    void setDescription(String description);

    Date getCreated();

    Date getUpdated();
    void setUpdated(Date updated);
    long getUpdateCount();
    void incrUpdateCount();

    Date getRemoved();
    void setRemoved(Date removed);
    String getRemovalReason();
    void setRemovalReason(String removalReason);
}
```

A.15 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/IDPSYNCPOLICYMANAGER.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
```

```
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy;

import com.cloud.user.UserAccount;
import com.cloud.utils.component.PluggableService;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.CreateIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.ListIdpSyncPoliciesCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.RemoveIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.UpdateIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.
    ↪ValidateIdpSyncPolicyMappingCmd;
import org.apache.cloudstack.api.response.IdpSyncPolicyMappingResponse;
import org.apache.cloudstack.framework.config.ConfigKey;

import java.util.List;
import java.util.Map;

public interface IdpSyncPolicyManager extends PluggableService {

    ConfigKey<Long> IdpSyncPolicyMappingTimeout = new ConfigKey<>("Advanced", Long.
    ↪class, "idp.sync.policy.mapping.timeout", "2000",
            "The maximum runtime, in milliseconds, to execute the IdP Sync Policy
    ↪mapping script; if it is reached, a timeout will happen.", true);

    ConfigKey<Boolean> IdpSyncPolicyAutoCreateDefaultPolicy = new ConfigKey<>("
    ↪Advanced", Boolean.class, "idp.sync.policy.auto.create.default.policy", "true",
            "Whether to create a default policy when an IdP is detected, if there is
    ↪ not already a policy with that IdP ID.", true);

    IdpSyncPolicy createIdpSyncPolicy(CreateIdpSyncPolicyCmd cmd);
    IdpSyncPolicy updateIdpSyncPolicy(UpdateIdpSyncPolicyCmd cmd);
    IdpSyncPolicy removeIdpSyncPolicy(RemoveIdpSyncPolicyCmd cmd);
    List<? extends IdpSyncPolicy> listIdpSyncPolicies(ListIdpSyncPoliciesCmd cmd);

    IdpSyncPolicyMappingResponse validateIdpSyncPolicyMapping(
    ↪ValidateIdpSyncPolicyMappingCmd cmd);
```

```
    UserAccount login(Map<String, Object> attributes, String idpId);

    void createDefaultPolicy(String idpId, String username);
}
```

A.16 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/MAPPING/ ACCOUNTMAP.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy.mapping;

import org.apache.cloudstack.utils.reflectiontostringbuilderutils.
    ↪ReflectionToStringBuilderUtils;

import java.util.Map;


public class AccountMap {
    private String accountname;
    private Map<String, String> details;
    private String networkdomain;
    private RoleMap role;
```

```
    private String uuid;

    public AccountMap() {
    }

    public String getAccountName() {
        return accountname;
    }

    public Map<String, String> getDetails() {
        return details;
    }

    public String getNetworkDomain() {
        return networkdomain;
    }

    public RoleMap getRole() {
        return role;
    }

    public String getUuid() {
        return uuid;
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilderUtils.reflectOnlySelectedFields(this, "uuid",
    ↪ "accountname, role");
    }
}
```

A.17 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/MAPPING/ATTRIBUTEMAP.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
```

```
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy.mapping;

import org.apache.commons.lang3.builder.ReflectionToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;

public class AttributeMap {
    private UserMap user;
    private AccountMap account;
    private DomainMap domain;
    private Boolean legacy;

    public AttributeMap(){
    }

    public UserMap getUser() {
        return user;
    }

    public AccountMap getAccount() {
        return account;
    }

    public DomainMap getDomain() {
        return domain;
    }
```

```
    public boolean shouldBypass() {
        return legacy != null && legacy && account == null && domain == null && user
    ↪ != null && user.hasOnlyUsername();
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilder.reflectionToString(this, ToStringStyle.
    ↪JSON_STYLE);
    }
}
```

A.18 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/MAPPING/DOMAINMAP.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy.mapping;

import org.apache.commons.lang3.builder.ReflectionToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;

public class DomainMap {
    private String path;
```

```
    private String uuid;

    public DomainMap() {
    }

    public String getPath() {
        return path;
    }

    public String getUuid() {
        return uuid;
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilder.reflectionToString(this, ToStringStyle.
    ↪JSON_STYLE);
    }
}
```

A.19 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/MAPPING/ROLEMAP.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
```

```
package org.apache.cloudstack.auth.idpsyncpolicy.mapping;

import org.apache.commons.lang3.builder.ReflectionToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;

public class RoleMap {
    private String name;
    private String type;
    private String uuid;

    public RoleMap() {
    }

    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

    public String getUuid() {
        return uuid;
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilder.reflectionToString(this, ToStringStyle.
    ↪JSON_STYLE);
    }
}
```

A.20 API/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/MAPPING/USERMAP.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
```

```
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy.mapping;

import org.apache.cloudstack.utils.reflectiontostringbuilderutils.
    ↪ReflectionToStringBuilderUtils;
import org.apache.commons.lang3.ObjectUtils;

public class UserMap {
    private String email;
    private String firstname;
    private String lastname;
    private String username;
    private String uuid;
    private String timezone;
    private String password;

    public UserMap() {
    }

    public String getEmail() {
        return email;
    }

    public String getFirstname() {
        return firstname;
    }
```

```
    public String getLastname() {
        return lastname;
    }

    public String getUsername() {
        return username;
    }

    public String getUuid() {
        return uuid;
    }

    public String getTimezone() {
        return timezone;
    }

    public String getPassword() {
        return password;
    }

    public boolean hasOnlyUsername() {
        return ObjectUtils.allNull(email, firstname, lastname, uuid, timezone) &&
    ↪username != null;
    }

    public void hidePassword() {
        this.password = null;
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilderUtils.reflectOnlySelectedFields(this, "uuid",
    ↪ "username");
    }
}
```

A.21 ENGINE/SCHEMA/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/IDPSYNCPOLICYVO.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy;


import com.cloud.utils.db.GenericDao;
import org.apache.cloudstack.utils.reflectiontostringbuilderutils.
    ↪ReflectionToStringBuilderUtils;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import java.util.Date;
```

```
import java.util.UUID;

@Entity
@Table(name = "idp_sync_policy")
public class IdpSyncPolicyVO implements IdpSyncPolicy {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private long id;

    @Column(name = "uuid", nullable = false)
    private String uuid = UUID.randomUUID().toString();

    @Column(name = "idp_id", nullable = false)
    private String idpId;

    @Column(name = "useraccount_operation", nullable = false)
    @Enumerated(EnumType.STRING)
    private Operation userAccountOperation = Operation.NONE;

    @Column(name = "mapping", nullable = false)
    private String mapping;

    @Column(name = "description")
    private String description;

    @Column(name = GenericDao.CREATED_COLUMN, nullable = false)
    @Temporal(value = TemporalType.TIMESTAMP)
    private Date created;

    @Column(name = "updated")
    @Temporal(value = TemporalType.TIMESTAMP)
    private Date updated;

    @Column(name = "update_count", nullable = false)
    private long updateCount;

    @Column(name = GenericDao.REMOVED_COLUMN)
    @Temporal(value = TemporalType.TIMESTAMP)
```

```
    private Date removed;

    @Column(name = "removal_reason")
    private String removalReason;

    public IdpSyncPolicyVO() {
    }

    public IdpSyncPolicyVO(String idpId, Operation operation, String mapping, String
    ↪ description) {
        this.idpId = idpId;
        this.userAccountOperation = operation;
        this.mapping = mapping;
        this.description = description;
    }

    @Override
    public long getId() {
        return id;
    }

    @Override
    public String getUuid() {
        return uuid;
    }

    @Override
    public String getIdpId() {
        return idpId;
    }

    @Override
    public Operation getOperation() {
        return userAccountOperation;
    }

    @Override
    public String getMapping() {
        return mapping;
```

```
}

@Override
public void setMapping(String mapping) {
    this.mapping = mapping;
}

@Override
public String getDescription() {
    return description;
}

@Override
public void setDescription(String description) {
    this.description = description;
}

@Override
public Date getCreated() {
    return created;
}

@Override
public void setUpdated(Date updated) {
    this.updated = updated;
}

@Override
public Date getUpdated() {
    return updated;
}

@Override
public long getUpdateCount() {
    return updateCount;
}

@Override
public void incrUpdateCount() {
```

```
        updateCount++;
    }

    @Override
    public Date getRemoved() {
        return removed;
    }

    @Override
    public void setRemoved(Date removed) {
        this.removed = removed;
    }

    @Override
    public String getRemovalReason() {
        return removalReason;
    }

    @Override
    public void setRemovalReason(String removalReason) {
        this.removalReason = removalReason;
    }

    @Override
    public String toString() {
        return ReflectionToStringBuilderUtils.reflectOnlySelectedFields(this, "uuid",
    ↪ "idpId", "useraccount_operation", "description");
    }
}
```

A.22 ENGINE/SCHEMA/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/DAO/IDPSYNCPOLICYDAO.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
```

```
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
package org.apache.cloudstack.auth.idpsyncpolicy.dao;

import com.cloud.utils.db.GenericDao;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyVO;

import java.util.List;


public interface IdpSyncPolicyDao extends GenericDao<IdpSyncPolicyVO, Long> {

    IdpSyncPolicyVO findActiveByIdpId(String idpId);
    IdpSyncPolicyVO findAnyByIdpId(String idpId);
    List<IdpSyncPolicyVO> listBy(Long id, String idpId, boolean showRemoved, Long
    ↪startIndex, Long pageSize, String keyword);

}
```

A.23 ENGINE/SCHEMA/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/DAO/IDPSYNCPOLICYDAOIMPL.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
```

```
//
// Unless required by applicable law or agreed to in writing,
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.
package org.apache.cloudstack.auth.idpsyncpolicy.dao;

import com.cloud.utils.db.Filter;
import com.cloud.utils.db.GenericDaoBase;
import com.cloud.utils.db.SearchBuilder;
import com.cloud.utils.db.SearchCriteria;
import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyVO;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class IdpSyncPolicyDaoImpl extends GenericDaoBase<IdpSyncPolicyVO, Long>
    ↪implements IdpSyncPolicyDao {

    private final SearchBuilder<IdpSyncPolicyVO> search;

    public IdpSyncPolicyDaoImpl() {
        search = createSearchBuilder();
        search.and("id", search.entity().getId(), SearchCriteria.Op.EQ);
        search.and("idp_id", search.entity().getIdpId(), SearchCriteria.Op.EQ);
        search.and("description", search.entity().getDescription(), SearchCriteria.
    ↪Op.LIKE);
        search.done();
    }

    @Override
    public IdpSyncPolicyVO findActiveByIdpId(String idpId) {
        SearchCriteria<IdpSyncPolicyVO> sc = search.create();
        sc.setParameters("idp_id", idpId);
        return findOneBy(sc);
    }
```

```
    @Override
    public IdpSyncPolicyVO findAnyByIdpId(String idpId) {
        SearchCriteria<IdpSyncPolicyVO> sc = search.create();
        sc.setParameters("idp_id", idpId);
        return findOneIncludingRemovedBy(sc);
    }

    @Override
    public List<IdpSyncPolicyVO> listBy(Long id, String idpId, boolean showRemoved,
    ↪Long startIndex, Long pageSize, String keyword) {
        SearchCriteria<IdpSyncPolicyVO> sc = search.create();
        sc.setParametersIfNotNull("id", id);
        sc.setParametersIfNotNull("idp_id", idpId);

        if (keyword != null) {
            sc.setParameters("description", "%" + keyword + "%");
        }

        Filter sorter = new Filter(IdpSyncPolicyVO.class, "created", true,
    ↪startIndex, pageSize);
        return (showRemoved) ? searchIncludingRemoved(sc, sorter, null, false) :
    ↪search(sc, sorter);
    }
}
```

A.24 ENGINE/SCHEMA/SRC/MAIN/RESOURCES/META-INF/CLOUDSTACK/CORE/SPRING-ENGINE-SCHEMA-CORE-DAOS-CONTEXT.XML

```
diff --git a/engine/schema/src/main/resources/META-INF/cloudstack/core/spring-engine-
    ↪schema-core-daos-context.xml b/engine/schema/src/main/resources/META-INF/
    ↪cloudstack/core/spring-engine-schema-core-daos-context.xml
index 171685ce41..78b5f84834 100644
--- a/engine/schema/src/main/resources/META-INF/cloudstack/core/spring-engine-schema-
    ↪core-daos-context.xml
+++ b/engine/schema/src/main/resources/META-INF/cloudstack/core/spring-engine-schema-
    ↪core-daos-context.xml
@@ -96,6 +96,7 @@
    <bean id="engineHostDetailsDaoImpl" class="org.apache.cloudstack.engine.
    ↪datacenter.entity.api.db.dao.HostDetailsDaoImpl" />
```

```
   <bean id="hostJoinDaoImpl" class="com.cloud.api.query.dao.HostJoinDaoImpl" />
   <bean id="engineHostTagsDaoImpl" class="org.apache.cloudstack.engine.datacenter.
   ↪entity.api.db.dao.HostTagsDaoImpl" />
+  <bean id="idpSyncPolicyDaoImpl" class="org.apache.cloudstack.auth.idpsyncpolicy.
   ↪dao.IdpSyncPolicyDaoImpl" />
   <bean id="iPAddressDaoImpl" class="com.cloud.network.dao.IPAddressDaoImpl" />
   <bean id="ip6GuestPrefixSubnetNetworkMapDaoImpl" class="com.cloud.network.dao.
   ↪Ipv6GuestPrefixSubnetNetworkMapDaoImpl" />
   <bean id="imageStoreDetailsDaoImpl" class="org.apache.cloudstack.storage.
   ↪datastore.db.ImageStoreDetailsDaoImpl" />
```

A.25 ENGINE/SCHEMA/SRC/MAIN/RESOURCES/META-INF/DB/SCHEMA-41910TO42000.SQL

```
diff --git a/engine/schema/src/main/resources/META-INF/db/schema-41910to42000.sql b/
   ↪engine/schema/src/main/resources/META-INF/db/schema-41910to42000.sql
index c36b71c2f2..0a307e19be 100644
--- a/engine/schema/src/main/resources/META-INF/db/schema-41910to42000.sql
+++ b/engine/schema/src/main/resources/META-INF/db/schema-41910to42000.sql
@@ -425,3 +425,20 @@ INSERT IGNORE INTO `cloud`.`guest_os_hypervisor` (uuid,
   ↪hypervisor_type, hypervi

 CALL `cloud`.`IDEMPOTENT_ADD_COLUMN`('cloud.vm_instance', 'delete_protection', '
   ↪boolean DEFAULT FALSE COMMENT "delete protection for vm" ');
 CALL `cloud`.`IDEMPOTENT_ADD_COLUMN`('cloud.volumes', 'delete_protection', 'boolean
   ↪ DEFAULT FALSE COMMENT "delete protection for volumes" ');
+
+-- IdP Sync Policy Table
+DROP TABLE IF EXISTS `cloud`.`idp_sync_policy`;
+CREATE TABLE `cloud`.`idp_sync_policy` (
+    `id` bigint unsigned NOT NULL AUTO_INCREMENT,
+    `uuid` varchar(40) UNIQUE NOT NULL,
+    `idp_id` text NOT NULL COMMENT 'URL of IdP',
+    `description` text,
+    `mapping` text NOT NULL COMMENT 'JavaScript script that maps attributes from
   ↪IdP to ACS entity',
+    `useraccount_operation` enum('NONE', 'CREATE', 'UPDATE', 'CREATEANDUPDATE') NOT
   ↪ NULL DEFAULT 'NONE' COMMENT 'Operation to be done with mapped attributes',
+    `created` datetime NOT NULL COMMENT 'Date in which policy was created',
+    `updated` datetime COMMENT 'Date in which policy was last updated, if not null',
```

```
    ↪
+    `update_count` bigint unsigned NOT NULL DEFAULT 0 COMMENT 'Number of times the
    ↪policy was updated',
+    `removed` datetime COMMENT 'Date in which policy was deleted, if not null',
+    `removal_reason` text COMMENT 'Reason for removing policy, if removed',
+    PRIMARY KEY (`id`), UNIQUE KEY(`uuid`)
+) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.26 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LDAPCREATEACCOUNTCMD.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/
    ↪api/command/LdapCreateAccountCmd.java b/plugins/user-authenticators/ldap/src/
    ↪main/java/org/apache/cloudstack/api/command/LdapCreateAccountCmd.java
index 880ecea4d1..01ae4d513f 100644
--- a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LdapCreateAccountCmd.java
+++ b/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LdapCreateAccountCmd.java
@@ -94,11 +94,11 @@ public class LdapCreateAccountCmd extends BaseCmd {
     UserAccount createCloudstackUserAccount(final LdapUser user, String accountName,
    ↪ Long domainId) {
         Account account = _accountService.getActiveAccountByName(accountName,
    ↪domainId);
         if (account == null) {
-            return _accountService.createUserAccount(username, generatePassword(),
    ↪user.getFirstname(), user.getLastname(), user.getEmail(), timezone, accountName
    ↪, getAccountType(), getRoleId(),
-                    domainId, networkDomain, details, accountUUID, userUUID, User.
    ↪Source.LDAP);
+            return _accountService.createUserAccount(username, generatePassword(),
    ↪user.getFirstname(), user.getLastname(), user.getEmail(), timezone, accountName
    ↪, getAccountType(),
+                    getRoleId(), domainId, networkDomain, details, accountUUID,
    ↪userUUID, null, User.Source.LDAP);
         } else {
             User newUser = _accountService.createUser(username, generatePassword(),
    ↪ user.getFirstname(), user.getLastname(), user.getEmail(), timezone,
    ↪accountName, domainId,
```

```
-                    userUUID, User.Source.LDAP);
+                    userUUID, null, User.Source.LDAP);
             return _accountService.getUserAccountById(newUser.getId());
         }
     }
```

A.27 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LDAPIMPORTUSERSCMD.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/
    ↪api/command/LdapImportUsersCmd.java b/plugins/user-authenticators/ldap/src/main
    ↪/java/org/apache/cloudstack/api/command/LdapImportUsersCmd.java
index 087bd63c29..e7c05143c8 100644
--- a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LdapImportUsersCmd.java
+++ b/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LdapImportUsersCmd.java
@@ -108,14 +108,14 @@ public class LdapImportUsersCmd extends BaseListCmd {
         if (account == null) {
             logger.debug("No account exists with name: " + accountName + " creating
    ↪ the account and an user with name: " + user.getUsername() + " in the account")
    ↪;
             _accountService.createUserAccount(user.getUsername(), generatePassword
    ↪(), user.getFirstname(), user.getLastname(), user.getEmail(), timezone,
    ↪accountName, getAccountType(), getRoleId(),
-                    domain.getId(), domain.getNetworkDomain(), details, UUID.
    ↪randomUUID().toString(), UUID.randomUUID().toString(), User.Source.LDAP);
+                    domain.getId(), domain.getNetworkDomain(), details, UUID.
    ↪randomUUID().toString(), UUID.randomUUID().toString(), null, User.Source.LDAP);
         } else {
 //            check if the user exists. if yes, call update
             UserAccount csuser = _accountService.getActiveUserAccount(user.
    ↪getUsername(), domain.getId());
             if (csuser == null) {
                 logger.debug("No user exists with name: " + user.getUsername() + "
    ↪creating a user in the account: " + accountName);
                 _accountService.createUser(user.getUsername(), generatePassword(),
    ↪user.getFirstname(), user.getLastname(), user.getEmail(), timezone, accountName
    ↪, domain.getId(),
```

```
-                        UUID.randomUUID().toString(), User.Source.LDAP);
+                        UUID.randomUUID().toString(), null, User.Source.LDAP);
             } else {
                 logger.debug("Account [name=%s] and user [name=%s] already exist in
    ↪ CloudStack. Executing the user update.");
```

A.28 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LINKACCOUNTTOLDAPCMD.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/
    ↪api/command/LinkAccountToLdapCmd.java b/plugins/user-authenticators/ldap/src/
    ↪main/java/org/apache/cloudstack/api/command/LinkAccountToLdapCmd.java
index 6219fc90f8..7100eac483 100644
--- a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LinkAccountToLdapCmd.java
+++ b/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
    ↪command/LinkAccountToLdapCmd.java
@@ -89,7 +89,7 @@ public class LinkAccountToLdapCmd extends BaseCmd {
                         try {
                             UserAccount userAccount = _accountService
                                     .createUserAccount(admin, "", ldapUser.
    ↪getFirstname(), ldapUser.getLastname(), ldapUser.getEmail(), null, admin,
    ↪Account.Type.DOMAIN_ADMIN, RoleType.DomainAdmin.getId(), domainId, null, null,
    ↪UUID.randomUUID().toString(),
-                                            UUID.randomUUID().toString(), User.
    ↪Source.LDAP);
+                                            UUID.randomUUID().toString(), null,
    ↪User.Source.LDAP);
                             response.setAdminId(String.valueOf(userAccount.
    ↪getAccountId()));
                             logger.info("created an account with name " + admin + "
    ↪ in the given domain " + domainId);
                         } catch (Exception e) {
```

A.29 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LINKDOMAINTOLDAPCMD.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/
```

```
   ↪api/command/LinkDomainToLdapCmd.java b/plugins/user-authenticators/ldap/src/
   ↪main/java/org/apache/cloudstack/api/command/LinkDomainToLdapCmd.java
index d5187f99c9..d0539f6620 100644
--- a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
   ↪command/LinkDomainToLdapCmd.java
+++ b/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/api/
   ↪command/LinkDomainToLdapCmd.java
@@ -105,7 +105,7 @@ public class LinkDomainToLdapCmd extends BaseCmd {
                     if (account == null) {
                         try {
                             UserAccount userAccount = _accountService.
   ↪createUserAccount(admin, "", ldapUser.getFirstname(), ldapUser.getLastname(),
   ↪ldapUser.getEmail(), null,
-                                    admin, Account.Type.DOMAIN_ADMIN, RoleType.
   ↪DomainAdmin.getId(), domainId, null, null, UUID.randomUUID().toString(), UUID.
   ↪randomUUID().toString(), User.Source.LDAP);
+                                    admin, Account.Type.DOMAIN_ADMIN, RoleType.
   ↪DomainAdmin.getId(), domainId, null, null, UUID.randomUUID().toString(), UUID.
   ↪randomUUID().toString(), null, User.Source.LDAP);
                             response.setAdminId(String.valueOf(userAccount.
   ↪getAccountId()));
                             logger.info("created an account with name " + admin + "
   ↪ in the given domain " + domainId);
                         } catch (Exception e) {
```

A.30 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/LDAP/LDAPAUTHENTICATOR.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/
   ↪ldap/LdapAuthenticator.java b/plugins/user-authenticators/ldap/src/main/java/
   ↪org/apache/cloudstack/ldap/LdapAuthenticator.java
index b850988159..d3d7463a64 100644
--- a/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/ldap/
   ↪LdapAuthenticator.java
+++ b/plugins/user-authenticators/ldap/src/main/java/org/apache/cloudstack/ldap/
   ↪LdapAuthenticator.java
@@ -149,7 +149,7 @@ public class LdapAuthenticator extends AdapterBase implements
   ↪UserAuthenticator
                 if (userAccount == null) {
```

```
                    // new user that is in ldap; authenticate and create
                    User user = _accountManager.createUser(username, "", ldapUser.
    ↪getFirstname(), ldapUser.getLastname(), ldapUser.getEmail(), null, accountName,
-                           domainId, UUID.randomUUID().toString(), User.Source.
    ↪LDAP);
+                           domainId, UUID.randomUUID().toString(), null, User.
    ↪Source.LDAP);
                    /* expected error conditions:
                     *
                     * caught in APIServlet: CloudRuntimeException("The domain " +
    ↪domainId + " does not exist; unable to create user");
@@ -296,7 +296,7 @@ public class LdapAuthenticator extends AdapterBase implements
    ↪UserAuthenticator
         String username = user.getUsername();
         _accountManager.createUserAccount(username, "", user.getFirstname(), user.
    ↪getLastname(), user.getEmail(), null, username,
                                           accountType, RoleType.getByAccountType(
    ↪accountType).getId(), domainId, null, null,
-                                          UUID.randomUUID().toString(), UUID.
    ↪randomUUID().toString(), User.Source.LDAP);
+                                          UUID.randomUUID().toString(), UUID.
    ↪randomUUID().toString(), null, User.Source.LDAP);
     }

     private void disableUserInCloudStack(UserAccount user) {
```

A.31 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/TEST/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LINKACCOUNTTOLDAPCMDTEST.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/
    ↪api/command/LinkAccountToLdapCmdTest.java b/plugins/user-authenticators/ldap/
    ↪src/test/java/org/apache/cloudstack/api/command/LinkAccountToLdapCmdTest.java
index 62a3a809b1..f5b08dd8ea 100644
--- a/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/api/
    ↪command/LinkAccountToLdapCmdTest.java
+++ b/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/api/
    ↪command/LinkAccountToLdapCmdTest.java
@@ -84,7 +84,7 @@ public class LinkAccountToLdapCmdTest implements
    ↪LdapConfigurationChanger {
```

```
         userAccount.setAccountId(24);
         when(accountService.createUserAccount(eq(username), eq(""), eq("Admin"), eq
    ↪("Admin"), eq("admin@ccp.citrix.com"), isNull(String.class),
                 eq(username), eq(Account.Type.DOMAIN_ADMIN), eq(RoleType.
    ↪DomainAdmin.getId()), eq(domainId), isNull(String.class),
-                (java.util.Map<String,String>)isNull(), anyString(), anyString(),
    ↪eq(User.Source.LDAP))).thenReturn(userAccount);
+                (java.util.Map<String,String>)isNull(), anyString(), anyString(),
    ↪null, eq(User.Source.LDAP))).thenReturn(userAccount);

         linkAccountToLdapCmd.execute();
         LinkAccountToLdapResponse result = (LinkAccountToLdapResponse)
    ↪linkAccountToLdapCmd.getResponseObject();
```

A.32 PLUGINS/USER-AUTHENTICATORS/LDAP/SRC/TEST/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/LINKDOMAINTOLDAPCMDTEST.JAVA

```
diff --git a/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/
    ↪api/command/LinkDomainToLdapCmdTest.java b/plugins/user-authenticators/ldap/src
    ↪/test/java/org/apache/cloudstack/api/command/LinkDomainToLdapCmdTest.java
index 67d0e77052..b35197ec20 100644
--- a/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/api/
    ↪command/LinkDomainToLdapCmdTest.java
+++ b/plugins/user-authenticators/ldap/src/test/java/org/apache/cloudstack/api/
    ↪command/LinkDomainToLdapCmdTest.java
@@ -82,7 +82,7 @@ public class LinkDomainToLdapCmdTest implements
    ↪LdapConfigurationChanger
         userAccount.setAccountId(24);
         when(accountService.createUserAccount(eq(username), eq(""), eq("Admin"), eq
    ↪("Admin"), eq("admin@ccp.citrix.com"), isNull(String.class),
                 eq(username), eq(Account.Type.DOMAIN_ADMIN), eq(RoleType.
    ↪DomainAdmin.getId()), eq(domainId), isNull(String.class),
-                (java.util.Map<String,String>)isNull(), anyString(), anyString(),
    ↪eq(User.Source.LDAP))).thenReturn(userAccount);
+                (java.util.Map<String,String>)isNull(), anyString(), anyString(),
    ↪null, eq(User.Source.LDAP))).thenReturn(userAccount);


         linkDomainToLdapCmd.execute();
```

A.33 PLUGINS/USER-AUTHENTICATORS/SAML2/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/SAML2LOGINAPIAUTHENTICATORCMD.JAVA

```
diff --git a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/
    ↪api/command/SAML2LoginAPIAuthenticatorCmd.java b/plugins/user-authenticators/
    ↪saml2/src/main/java/org/apache/cloudstack/api/command/
    ↪SAML2LoginAPIAuthenticatorCmd.java
index 742680fdcc..b3284570bd 100644
--- a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/api/
    ↪command/SAML2LoginAPIAuthenticatorCmd.java
+++ b/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/api/
    ↪command/SAML2LoginAPIAuthenticatorCmd.java
@@ -18,6 +18,8 @@ package org.apache.cloudstack.api.command;

 import java.io.IOException;
 import java.net.InetAddress;
+import java.security.cert.X509Certificate;
+import java.util.HashMap;
 import java.util.List;
 import java.util.Map;

@@ -39,6 +41,7 @@ import org.apache.cloudstack.api.auth.APIAuthenticationType;
 import org.apache.cloudstack.api.auth.APIAuthenticator;
 import org.apache.cloudstack.api.auth.PluggableAPIAuthenticator;
 import org.apache.cloudstack.api.response.LoginCmdResponse;
+import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;
 import org.apache.cloudstack.framework.config.ConfigKey;
 import org.apache.cloudstack.framework.config.Configurable;
 import org.apache.cloudstack.saml.SAML2AuthManager;
@@ -74,9 +77,6 @@ import com.cloud.exception.CloudAuthenticationException;
 import com.cloud.user.Account;
 import com.cloud.user.DomainManager;
 import com.cloud.user.UserAccount;
-import com.cloud.user.UserAccountVO;
-import com.cloud.user.dao.UserAccountDao;
-import com.cloud.utils.db.EntityManager;

 @APICommand(name = "samlSso", description = "SP initiated SAML Single Sign On",
```

```
   ↪requestHasSensitiveInfo = true, responseObject = LoginCmdResponse.class,
   ↪entityType = {})
 public class SAML2LoginAPIAuthenticatorCmd extends BaseCmd implements
   ↪APIAuthenticator, Configurable {
@@ -91,11 +91,9 @@ public class SAML2LoginAPIAuthenticatorCmd extends BaseCmd
   ↪implements APIAuthent
     @Inject
     ApiServerService apiServer;
     @Inject
-    EntityManager entityMgr;
-    @Inject
     DomainManager domainMgr;
     @Inject
-    private UserAccountDao userAccountDao;
+    IdpSyncPolicyManager idpSyncPolicyManager;

     protected static ConfigKey<String> saml2FailedLoginRedirectUrl = new ConfigKey<
   ↪String>("Advanced", String.class, "saml2.failed.login.redirect.url", "",
             "The URL to redirect the SAML2 login failed message (the default vaulue
   ↪ is empty).", true);
@@ -142,198 +140,184 @@ public class SAML2LoginAPIAuthenticatorCmd extends BaseCmd
   ↪implements APIAuthent
         return responseObject;
     }

-    protected void checkAndFailOnMissingSAMLSignature(Signature signature) {
-        if (signature == null && SAML2AuthManager.SAMLCheckSignature.value()) {
+    protected void validateSignature(final Map<String, Object[]> params, String
   ↪responseType, Signature sig, X509Certificate certificate) {
+        if (sig == null && SAML2AuthManager.SAMLCheckSignature.value()) {
             logger.error("Failing SAML login due to missing signature in the SAML
   ↪response and signature check is enforced. " +
                     "Please check and ensure the IDP configuration has signing
   ↪certificate or relax the saml2.check.signature setting.");
             throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Signature is
   ↪ missing from the SAML Response. Please contact the Administrator");
         }
+        if (certificate == null || sig == null) {
+            return;
```

```
+        }
+        BasicX509Credential credential = new BasicX509Credential();
+        credential.setEntityCertificate(certificate);
+        SignatureValidator validator = new SignatureValidator(credential);
+        try {
+            validator.validate(sig);
+        } catch (ValidationException e) {
+            logger.error("SAML Response's signature failed to be validated by IDP
    ↪signing key: {}", e.getMessage());
+            throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
+                    "SAML Response's signature failed to be validated by IDP
    ↪signing key",
+                    params, responseType));
+        }
     }

-    @Override
-    public String authenticate(final String command, final Map<String, Object[]>
    ↪params, final HttpSession session, final InetAddress remoteAddress, final
    ↪String responseType, final StringBuilder auditTrailSb, final HttpServletRequest
    ↪ req, final HttpServletResponse resp) throws ServerApiException {
-        try {
-            if (!params.containsKey(SAMLPluginConstants.SAML_RESPONSE) && !params.
    ↪containsKey("SAMLart")) {
-                String idpId = null;
-                String domainPath = null;
+    private void makeSamlRequest(final Map<String, Object[]> params, final String
    ↪responseType, final HttpServletResponse resp) throws IOException {
+        String idpId = null;
+        String domainPath = null;

-                if (params.containsKey(ApiConstants.IDP_ID)) {
-                    idpId = ((String[])params.get(ApiConstants.IDP_ID))[0];
-                }
+        if (params.containsKey(ApiConstants.IDP_ID)) {
+            idpId = ((String[])params.get(ApiConstants.IDP_ID))[0];
+        }
```

```
-                if (params.containsKey(ApiConstants.DOMAIN)) {
-                    domainPath = ((String[])params.get(ApiConstants.DOMAIN))[0];
-                }
+        if (params.containsKey(ApiConstants.DOMAIN)) {
+            domainPath = ((String[])params.get(ApiConstants.DOMAIN))[0];
+        }

-                if (domainPath != null && !domainPath.isEmpty()) {
-                    if (!domainPath.startsWith("/")) {
-                        domainPath = "/" + domainPath;
-                    }
-                    if (!domainPath.endsWith("/")) {
-                        domainPath = domainPath + "/";
-                    }
-                }
+        if (domainPath != null && !domainPath.isEmpty()) {
+            if (!domainPath.startsWith("/")) {
+                domainPath = "/" + domainPath;
+            }
+            if (!domainPath.endsWith("/")) {
+                domainPath = domainPath + "/";
+            }
+        }

-                SAMLProviderMetadata spMetadata = samlAuthManager.getSPMetadata();
-                SAMLProviderMetadata idpMetadata = samlAuthManager.getIdPMetadata(
    ↪idpId);
-                if (idpMetadata == null) {
-                    throw new ServerApiException(ApiErrorCode.PARAM_ERROR,
    ↪apiServer.getSerializedApiError(ApiErrorCode.PARAM_ERROR.getHttpCode(),
-                            "IdP ID (" + idpId + ") is not found in our list of
    ↪supported IdPs, cannot proceed.",
-                            params, responseType));
-                }
-                if (idpMetadata.getSsoUrl() == null || idpMetadata.getSsoUrl().
    ↪isEmpty()) {
-                    throw new ServerApiException(ApiErrorCode.PARAM_ERROR,
    ↪apiServer.getSerializedApiError(ApiErrorCode.PARAM_ERROR.getHttpCode(),
-                            "IdP ID (" + idpId + ") has no Single Sign On URL
```

```
    ↪defined please contact "
-                                   + idpMetadata.getContactPersonName() + " <" +
    ↪idpMetadata.getContactPersonEmail() + ">, cannot proceed.",
-                           params, responseType));
-               }
-               String authnId = SAMLUtils.generateSecureRandomId();
-               samlAuthManager.saveToken(authnId, domainPath, idpMetadata.
    ↪getEntityId());
-               logger.debug("Sending SAMLRequest id=" + authnId);
-               String redirectUrl = SAMLUtils.buildAuthnRequestUrl(authnId,
    ↪spMetadata, idpMetadata, SAML2AuthManager.SAMLSignatureAlgorithm.value());
-               resp.sendRedirect(redirectUrl);
-               return "";
-           } if (params.containsKey("SAMLart")) {
+       SAMLProviderMetadata spMetadata = samlAuthManager.getSPMetadata();
+       SAMLProviderMetadata idpMetadata = samlAuthManager.getIdPMetadata(idpId);
+       if (idpMetadata == null) {
+           throw new ServerApiException(ApiErrorCode.PARAM_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.PARAM_ERROR.getHttpCode(),
+                   "IdP ID (" + idpId + ") is not found in our list of supported
    ↪IdPs, cannot proceed.",
+                   params, responseType));
+       }
+       if (idpMetadata.getSsoUrl() == null || idpMetadata.getSsoUrl().isEmpty()) {
+           throw new ServerApiException(ApiErrorCode.PARAM_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.PARAM_ERROR.getHttpCode(),
+                   "IdP ID (" + idpId + ") has no Single Sign On URL defined
    ↪please contact "
+                           + idpMetadata.getContactPersonName() + " <" +
    ↪idpMetadata.getContactPersonEmail() + ">, cannot proceed.",
+                   params, responseType));
+       }
+       String authnId = SAMLUtils.generateSecureRandomId();
+       samlAuthManager.saveToken(authnId, domainPath, idpMetadata.getEntityId());
+       logger.debug("Sending SAMLRequest id={}", authnId);
+       String redirectUrl = SAMLUtils.buildAuthnRequestUrl(authnId, spMetadata,
    ↪idpMetadata, SAML2AuthManager.SAMLSignatureAlgorithm.value());
+       resp.sendRedirect(redirectUrl);
+    }
```

```
+
+    private void validateAndPurgeToken(final Map<String, Object[]> params, String
    ↪responseType, Issuer issuer, Response processedSAMLResponse) {
+        String responseToId = processedSAMLResponse.getInResponseTo();
+        logger.debug("Received SAMLResponse in response to id={}", responseToId);
+        SAMLTokenVO token = samlAuthManager.getToken(responseToId);
+        if (token != null) {
+            if (!(token.getEntity().equalsIgnoreCase(issuer.getValue()))) {
+                throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
+                        "The SAML response contains Issuer Entity ID that is
    ↪different from the original SAML request",
+                        params, responseType));
+            }
+        } else {
+            throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
+                    "Received SAML response for a SSO request that we may not have
    ↪made or has expired, please try logging in again",
+                    params, responseType));
+        }
+        samlAuthManager.purgeToken(token);
+    }
+
+    @Override
+    public String authenticate(final String command, final Map<String, Object[]>
    ↪params, final HttpSession session, final InetAddress remoteAddress, final
    ↪String responseType,
+                               final StringBuilder auditTrailSb, final
    ↪HttpServletRequest req, final HttpServletResponse resp) throws
    ↪ServerApiException {
+        try {
+            if (params.containsKey("SAMLart")) {
                throw new ServerApiException(ApiErrorCode.UNSUPPORTED_ACTION_ERROR,
    ↪ apiServer.getSerializedApiError(ApiErrorCode.UNSUPPORTED_ACTION_ERROR.
    ↪getHttpCode(),
-                       "SAML2 HTTP Artifact Binding is not supported",
+                       "SAML2 HTTP Artifact Binding is not supported", params,
    ↪responseType));
```

```
+            }
+
+            if (!params.containsKey(SAMLPluginConstants.SAML_RESPONSE)) {
+                makeSamlRequest(params, responseType, resp);
+                return "";
+            }
+
+            final String samlResponse = ((String[])params.get(SAMLPluginConstants.
    ↪SAML_RESPONSE))[0];
+            Response processedSAMLResponse = this.processSAMLResponse(samlResponse)
    ↪;
+            String statusCode = processedSAMLResponse.getStatus().getStatusCode().
    ↪getValue();
+            if (!statusCode.equals(StatusCode.SUCCESS_URI)) {
+                throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
+                        "Identity Provider send a non-successful authentication
    ↪status code",
                         params, responseType));
-            } else {
-                final String samlResponse = ((String[])params.get(
    ↪SAMLPluginConstants.SAML_RESPONSE))[0];
-                Response processedSAMLResponse = this.processSAMLResponse(
    ↪samlResponse);
-                String statusCode = processedSAMLResponse.getStatus().getStatusCode
    ↪().getValue();
-                if (!statusCode.equals(StatusCode.SUCCESS_URI)) {
-                    throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR,
    ↪apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
-                            "Identity Provider send a non-successful authentication
    ↪ status code",
-                            params, responseType));
-                }
+            }

-                String username = null;
-                Issuer issuer = processedSAMLResponse.getIssuer();
-                SAMLProviderMetadata spMetadata = samlAuthManager.getSPMetadata();
-                SAMLProviderMetadata idpMetadata = samlAuthManager.getIdPMetadata(
```

```
   ↪issuer.getValue());
-
-               String responseToId = processedSAMLResponse.getInResponseTo();
-               logger.debug("Received SAMLResponse in response to id=" +
   ↪responseToId);
-               SAMLTokenVO token = samlAuthManager.getToken(responseToId);
-               if (token != null) {
-                   if (!(token.getEntity().equalsIgnoreCase(issuer.getValue()))) {
-                       throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR,
   ↪apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
-                               "The SAML response contains Issuer Entity ID that
   ↪is different from the original SAML request",
-                               params, responseType));
-                   }
-               } else {
-                   throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR,
   ↪apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
-                           "Received SAML response for a SSO request that we may
   ↪not have made or has expired, please try logging in again",
-                           params, responseType));
-               }
-               samlAuthManager.purgeToken(token);
+           Issuer issuer = processedSAMLResponse.getIssuer();
+           SAMLProviderMetadata spMetadata = samlAuthManager.getSPMetadata();
+           SAMLProviderMetadata idpMetadata = samlAuthManager.getIdPMetadata(
   ↪issuer.getValue());

-               // Set IdpId for this session
-               session.setAttribute(SAMLPluginConstants.SAML_IDPID, issuer.
   ↪getValue());
+           validateAndPurgeToken(params, responseType, issuer,
   ↪processedSAMLResponse);

-               Signature sig = processedSAMLResponse.getSignature();
-               checkAndFailOnMissingSAMLSignature(sig);
-               if (idpMetadata.getSigningCertificate() != null && sig != null) {
-                   BasicX509Credential credential = new BasicX509Credential();
-                   credential.setEntityCertificate(idpMetadata.
   ↪getSigningCertificate());
```

```
-                    SignatureValidator validator = new SignatureValidator(
    ↪credential);
-                    try {
-                        validator.validate(sig);
-                    } catch (ValidationException e) {
-                        logger.error("SAML Response's signature failed to be
    ↪validated by IDP signing key:" + e.getMessage());
-                        throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR,
    ↪apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
-                                "SAML Response's signature failed to be validated
    ↪by IDP signing key",
-                                params, responseType));
-                    }
-                }
+            // Set IdpId for this session
+            session.setAttribute(SAMLPluginConstants.SAML_IDPID, issuer.getValue())
    ↪;

-                username = SAMLUtils.getValueFromAssertions(processedSAMLResponse.
    ↪getAssertions(), SAML2AuthManager.SAMLUserAttributeName.value());
+            Signature sig = processedSAMLResponse.getSignature();
+            validateSignature(params, responseType, sig, idpMetadata.
    ↪getSigningCertificate());

-                for (Assertion assertion: processedSAMLResponse.getAssertions()) {
-                    if (assertion!= null && assertion.getSubject() != null &&
    ↪assertion.getSubject().getNameID() != null) {
-                        session.setAttribute(SAMLPluginConstants.SAML_NAMEID,
    ↪assertion.getSubject().getNameID().getValue());
-                        break;
-                    }
+            Map<String, Object> attributes = new HashMap<>();
+            String samlNameId = null;
+            for (Assertion assertion: processedSAMLResponse.getAssertions()) {
+                if (samlNameId == null && assertion.getSubject() != null &&
    ↪assertion.getSubject().getNameID() != null) {
+                    samlNameId = assertion.getSubject().getNameID().getValue();
                }
+                attributes = SAMLUtils.collectAssertionAttributes(attributes,
```

```
    ↪assertion);
+           }

-               if (idpMetadata.getEncryptionCertificate() != null && spMetadata !=
    ↪ null
-                       && spMetadata.getKeyPair() != null && spMetadata.getKeyPair
    ↪().getPrivate() != null) {
-                   Credential credential = SecurityHelper.getSimpleCredential(
    ↪idpMetadata.getEncryptionCertificate().getPublicKey(),
-                           spMetadata.getKeyPair().getPrivate());
-                   StaticKeyInfoCredentialResolver keyInfoResolver = new
    ↪StaticKeyInfoCredentialResolver(credential);
-                   ChainingEncryptedKeyResolver keyResolver = new
    ↪ChainingEncryptedKeyResolver();
-                   keyResolver.getResolverChain().add(new
    ↪InlineEncryptedKeyResolver());
-                   keyResolver.getResolverChain().add(new
    ↪EncryptedElementTypeEncryptedKeyResolver());
-                   Decrypter decrypter = new Decrypter(null, keyInfoResolver,
    ↪keyResolver);
-                   decrypter.setRootInNewDocument(true);
-                   List<EncryptedAssertion> encryptedAssertions =
    ↪processedSAMLResponse.getEncryptedAssertions();
-                   if (encryptedAssertions != null) {
-                       for (EncryptedAssertion encryptedAssertion :
    ↪encryptedAssertions) {
-                           Assertion assertion = null;
-                           try {
-                               assertion = decrypter.decrypt(encryptedAssertion);
-                           } catch (DecryptionException e) {
-                               logger.warn("SAML EncryptedAssertion error: " + e.
    ↪toString());
-                           }
-                           if (assertion == null) {
-                               continue;
-                           }
-                           Signature encSig = assertion.getSignature();
-                           checkAndFailOnMissingSAMLSignature(encSig);
-                           if (idpMetadata.getSigningCertificate() != null &&
```

```
    ↪encSig != null) {
-                               BasicX509Credential sigCredential = new
    ↪BasicX509Credential();
-                               sigCredential.setEntityCertificate(idpMetadata.
    ↪getSigningCertificate());
-                               SignatureValidator validator = new
    ↪SignatureValidator(sigCredential);
-                               try {
-                                   validator.validate(encSig);
-                               } catch (ValidationException e) {
-                                   logger.error("SAML Response's signature failed
    ↪to be validated by IDP signing key:" + e.getMessage());
-                                   throw new ServerApiException(ApiErrorCode.
    ↪ACCOUNT_ERROR, apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.
    ↪getHttpCode(),
-                                           "SAML Response's signature failed to be
    ↪ validated by IDP signing key",
-                                           params, responseType));
-                               }
-                           }
-                           if (assertion.getSubject() != null && assertion.
    ↪getSubject().getNameID() != null) {
-                               session.setAttribute(SAMLPluginConstants.
    ↪SAML_NAMEID, assertion.getSubject().getNameID().getValue());
-                           }
-                           if (username == null) {
-                               username = SAMLUtils.
    ↪getValueFromAttributeStatements(assertion.getAttributeStatements(),
    ↪SAML2AuthManager.SAMLUserAttributeName.value());
-                           }
-                       }
+           List<EncryptedAssertion> encryptedAssertions = processedSAMLResponse.
    ↪getEncryptedAssertions();
+           if (idpMetadata.getEncryptionCertificate() != null && spMetadata !=
    ↪null && spMetadata.getKeyPair() != null &&
+                   spMetadata.getKeyPair().getPrivate() != null && !
    ↪encryptedAssertions.isEmpty()) {
+               Credential credential = SecurityHelper.getSimpleCredential(
    ↪idpMetadata.getEncryptionCertificate().getPublicKey(),
```

```
+                        spMetadata.getKeyPair().getPrivate());
+                StaticKeyInfoCredentialResolver keyInfoResolver = new
    ↪StaticKeyInfoCredentialResolver(credential);
+                ChainingEncryptedKeyResolver keyResolver = new
    ↪ChainingEncryptedKeyResolver();
+                keyResolver.getResolverChain().add(new InlineEncryptedKeyResolver()
    ↪);
+                keyResolver.getResolverChain().add(new
    ↪EncryptedElementTypeEncryptedKeyResolver());
+                Decrypter decrypter = new Decrypter(null, keyInfoResolver,
    ↪keyResolver);
+                decrypter.setRootInNewDocument(true);
+
+                for (EncryptedAssertion encryptedAssertion : encryptedAssertions) {
+                    Assertion assertion = null;
+                    try {
+                        assertion = decrypter.decrypt(encryptedAssertion);
+                    } catch (DecryptionException e) {
+                        logger.warn("SAML EncryptedAssertion error: {}", e);
+                    }
+                    if (assertion == null) {
+                        continue;
                     }
-                }

-                if (username == null) {
-                    throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR,
    ↪apiServer.getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
-                            "Failed to find admin configured username attribute in
    ↪the SAML Response. Please ask your administrator to check SAML user attribute
    ↪name.", params, responseType));
-                }
+                    Signature encSig = assertion.getSignature();
+                    validateSignature(params, responseType, encSig, idpMetadata.
    ↪getSigningCertificate());

-                UserAccount userAccount = null;
-                List<UserAccountVO> possibleUserAccounts = userAccountDao.
    ↪getAllUsersByNameAndEntity(username, issuer.getValue());
```

```
-                if (possibleUserAccounts != null && possibleUserAccounts.size() >
    ↪0) {
-                    // Log into the first enabled user account
-                    // Users can switch to other allowed accounts later
-                    for (UserAccountVO possibleUserAccount : possibleUserAccounts)
    ↪{
-                        if (possibleUserAccount.getAccountState().equals(Account.
    ↪State.ENABLED.toString())) {
-                            userAccount = possibleUserAccount;
-                            break;
-                        }
+                    if (samlNameId == null && assertion.getSubject() != null &&
    ↪assertion.getSubject().getNameID() != null) {
+                        samlNameId = assertion.getSubject().getNameID().getValue();
                     }
+                    attributes = SAMLUtils.collectAssertionAttributes(attributes,
    ↪assertion);
                }
+            }

-                whenFailToAuthenticateThrowExceptionOrRedirectToUrl(params,
    ↪responseType, resp, issuer, userAccount);
+            session.setAttribute(SAMLPluginConstants.SAML_NAMEID, samlNameId);

-                try {
-                    if (apiServer.verifyUser(userAccount.getId())) {
-                        LoginCmdResponse loginResponse = (LoginCmdResponse)
    ↪apiServer.loginUser(session, userAccount.getUsername(), userAccount.getUsername
    ↪() + userAccount.getSource().toString(),
-                                userAccount.getDomainId(), null, remoteAddress,
    ↪params);
-                        SAMLUtils.setupSamlUserCookies(loginResponse, resp);
-                        resp.sendRedirect(SAML2AuthManager.
    ↪SAMLCloudStackRedirectionUrl.value());
-                        return ApiResponseSerializer.toSerializedString(
    ↪loginResponse, responseType);
-                    }
-                } catch (CloudAuthenticationException | IOException exception) {
-                    logger.debug("SAML Login failed to log in the user due to: " +
```

```
    ↪exception.getMessage());
+           if (attributes.isEmpty()) {
+               throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, apiServer.
    ↪getSerializedApiError(ApiErrorCode.ACCOUNT_ERROR.getHttpCode(),
+                       "Failed to find any attributes in the SAML Response. Please
    ↪ contact your administrator.", params, responseType));
+           }
+
+           UserAccount userAccount = idpSyncPolicyManager.login(attributes, issuer.
    ↪getValue());
+           whenFailToAuthenticateThrowExceptionOrRedirectToUrl(params,
    ↪responseType, resp, issuer, userAccount);
+
+           try {
+               if (apiServer.verifyUser(userAccount.getId())) {
+                   LoginCmdResponse loginResponse = (LoginCmdResponse) apiServer.
    ↪loginUser(session, userAccount.getUsername(), userAccount.getUsername() +
    ↪userAccount.getSource().toString(),
+                           userAccount.getDomainId(), null, remoteAddress, params)
    ↪;
+                   SAMLUtils.setupSamlUserCookies(loginResponse, resp);
+                   resp.sendRedirect(SAML2AuthManager.SAMLCloudStackRedirectionUrl.
    ↪value());
+                   return ApiResponseSerializer.toSerializedString(loginResponse,
    ↪responseType);
                }
+           } catch (CloudAuthenticationException | IOException exception) {
+               logger.debug("SAML Login failed to log in the user due to: " +
    ↪exception.getMessage());
            }
        } catch (IOException e) {
            auditTrailSb.append("SP initiated SAML authentication using HTTP
    ↪redirection failed:");
```

A.34 PLUGINS/USER-AUTHENTICATORS/SAML2/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/SAML/SAML2AUTHMANAGERIMPL.JAVA

```
diff --git a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/
    ↪saml/SAML2AuthManagerImpl.java b/plugins/user-authenticators/saml2/src/main/
```

```
   ↪java/org/apache/cloudstack/saml/SAML2AuthManagerImpl.java
index 545b01a4a3..b7de0d1955 100644
--- a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
   ↪SAML2AuthManagerImpl.java
+++ b/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
   ↪SAML2AuthManagerImpl.java
@@ -50,6 +50,7 @@ import org.apache.cloudstack.api.command.ListIdpsCmd;
 import org.apache.cloudstack.api.command.ListSamlAuthorizationCmd;
 import org.apache.cloudstack.api.command.SAML2LoginAPIAuthenticatorCmd;
 import org.apache.cloudstack.api.command.SAML2LogoutAPIAuthenticatorCmd;
+import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;
 import org.apache.cloudstack.framework.config.ConfigKey;
 import org.apache.cloudstack.framework.config.Configurable;
 import org.apache.cloudstack.framework.security.keystore.KeystoreDao;
@@ -118,6 +119,9 @@ public class SAML2AuthManagerImpl extends AdapterBase implements
   ↪ SAML2AuthManage
     @Inject
     DomainManager _domainMgr;

+    @Inject
+    IdpSyncPolicyManager idpSyncPolicyManager;
+
     @Override
     public boolean start() {
         if (isSAMLPluginEnabled()) {
@@ -316,6 +320,7 @@ public class SAML2AuthManagerImpl extends AdapterBase implements
   ↪ SAML2AuthManage
                 logger.warn("Duplicate IdP metadata found with entity Id: " +
   ↪idpMetadata.getEntityId());
             }
             idpMap.put(idpMetadata.getEntityId(), idpMetadata);
+            idpSyncPolicyManager.createDefaultPolicy(idpMetadata.getEntityId(),
   ↪SAMLUserAttributeName.value());
         }
     }
```

A.35 PLUGINS/USER-AUTHENTICATORS/SAML2/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/SAML/SAML2USERAUTHENTICATOR.JAVA

```
diff --git a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/
    ↪saml/SAML2UserAuthenticator.java b/plugins/user-authenticators/saml2/src/main/
    ↪java/org/apache/cloudstack/saml/SAML2UserAuthenticator.java
index 6f9854ad17..94abcd70ba 100644
--- a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
    ↪SAML2UserAuthenticator.java
+++ b/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
    ↪SAML2UserAuthenticator.java
@@ -37,18 +37,16 @@ public class SAML2UserAuthenticator extends AdapterBase
    ↪implements UserAuthentic

     @Override
     public Pair<Boolean, ActionOnFailedAuthentication> authenticate(String username,
    ↪ String password, Long domainId, Map<String, Object[]> requestParameters) {
-        if (logger.isDebugEnabled()) {
-            logger.debug("Trying SAML2 auth for user: " + username);
-        }
+        logger.debug("Trying SAML2 auth for user [{}]. ", username);

-        if (StringUtils.isEmpty(username) || StringUtils.isEmpty(password)) {
-            logger.debug("Username or Password cannot be empty");
+        if (StringUtils.isEmpty(username)) {
+            logger.debug("Username cannot be empty");
             return new Pair<Boolean, ActionOnFailedAuthentication>(false, null);
         }

         final UserAccount userAccount = _userAccountDao.getUserAccount(username,
    ↪domainId);
         if (userAccount == null || userAccount.getSource() != User.Source.SAML2) {
-            logger.debug("Unable to find user with " + username + " in domain " +
    ↪domainId + ", or user source is not SAML2");
+            logger.debug("Unable to find user [{]] in domain [{}], or user source
    ↪is not SAML2.", username, domainId);
             return new Pair<Boolean, ActionOnFailedAuthentication>(false, null);
         } else {
             User user = _userDao.getUser(userAccount.getId());
```

A.36 PLUGINS/USER-AUTHENTICATORS/SAML2/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/SAML/SAMLUTILS.JAVA

```
diff --git a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/
   ↪saml/SAMLUtils.java b/plugins/user-authenticators/saml2/src/main/java/org/
   ↪apache/cloudstack/saml/SAMLUtils.java
index 54f6e84fe3..f8b2950a83 100644
--- a/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
   ↪SAMLUtils.java
+++ b/plugins/user-authenticators/saml2/src/main/java/org/apache/cloudstack/saml/
   ↪SAMLUtils.java
@@ -47,6 +47,7 @@ import java.security.spec.X509EncodedKeySpec;
 import java.util.HashMap;
 import java.util.List;
 import java.util.Map;
+import java.util.stream.Collectors;
 import java.util.zip.Deflater;
 import java.util.zip.DeflaterOutputStream;

@@ -118,38 +119,31 @@ public class SAMLUtils {
                 String.valueOf(charset.charAt(new SecureRandom().nextInt(charset.
   ↪length()))));
     }

-    public static String getValueFromAttributeStatements(final List<
   ↪AttributeStatement> attributeStatements, final String attributeKey) {
-        if (attributeStatements == null || attributeStatements.size() < 1 ||
   ↪attributeKey == null) {
-            return null;
+    /**
+     * Collects values from attributes of given assertion.
+     * @return Map of attribute name to value. If attribute is multivalued, map
   ↪entry's value is a list.
+     */
+    public static Map<String, Object> collectAssertionAttributes(Map<String, Object
   ↪> attributes, final Assertion assertion) {
+        if (attributes == null || assertion == null) {
+            return attributes;
```

```
         }
-        for (AttributeStatement attributeStatement : attributeStatements) {
-            if (attributeStatement == null || attributeStatements.size() < 1) {
-                continue;
-            }
-            for (Attribute attribute : attributeStatement.getAttributes()) {
-                if (attribute.getAttributeValues() != null && attribute.
    ↪getAttributeValues().size() > 0) {
-                    String value = attribute.getAttributeValues().get(0).getDOM().
    ↪getTextContent();
-                    LOGGER.debug("SAML attribute name: " + attribute.getName() + "
    ↪friendly-name:" + attribute.getFriendlyName() + " value:" + value);
-                    if (attributeKey.equals(attribute.getName()) || attributeKey.
    ↪equals(attribute.getFriendlyName())) {
-                        return value;
-                    }
+        for (AttributeStatement statement : assertion.getAttributeStatements()) {
+            for (Attribute attribute : statement.getAttributes()) {
+                List<XMLObject> attributeValues = attribute.getAttributeValues();
+                if (attributeValues == null || attributeValues.isEmpty()) {
+                    continue;
+                }
+                if (attributeValues.size() == 1) {
+                    attributes.put(attribute.getName(), attributeValues.get(0).
    ↪getDOM().getTextContent());
+                } else {
+                    List<String> values = attributeValues.stream()
+                            .map(value -> value.getDOM().getTextContent())
+                            .collect(Collectors.toList());
+                    attributes.put(attribute.getName(), values);
                 }
             }
         }
-        return null;
-    }
-
-    public static String getValueFromAssertions(final List<Assertion> assertions,
    ↪final String attributeKey) {
-        if (assertions == null || attributeKey == null) {
```

```
-            return null;
-        }
-        for (Assertion assertion : assertions) {
-            String value = getValueFromAttributeStatements(assertion.
    ↪getAttributeStatements(), attributeKey);
-            if (value != null) {
-                return value;
-            }
-        }
-        return null;
+        return attributes;
     }

     public static String buildAuthnRequestUrl(final String authnId, final
    ↪SAMLProviderMetadata spMetadata, final SAMLProviderMetadata idpMetadata, final
    ↪String signatureAlgorithm) {
```

A.37 PLUGINS/USER-AUTHENTICATORS/SAML2/SRC/TEST/JAVA/ORG/APACHE/CLOUDSTACK/API/COMMAND/SAML2LOGINAPIAUTHENTICATORCMDTEST.JAVA

```
diff --git a/plugins/user-authenticators/saml2/src/test/java/org/apache/cloudstack/
    ↪api/command/SAML2LoginAPIAuthenticatorCmdTest.java b/plugins/user-
    ↪authenticators/saml2/src/test/java/org/apache/cloudstack/api/command/
    ↪SAML2LoginAPIAuthenticatorCmdTest.java
index 48a3139052..fb18ccccb4 100644
--- a/plugins/user-authenticators/saml2/src/test/java/org/apache/cloudstack/api/
    ↪command/SAML2LoginAPIAuthenticatorCmdTest.java
+++ b/plugins/user-authenticators/saml2/src/test/java/org/apache/cloudstack/api/
    ↪command/SAML2LoginAPIAuthenticatorCmdTest.java
@@ -38,6 +38,7 @@ import org.apache.cloudstack.api.ApiServerService;
 import org.apache.cloudstack.api.BaseCmd;
 import org.apache.cloudstack.api.ServerApiException;
 import org.apache.cloudstack.api.auth.APIAuthenticationType;
+import org.apache.cloudstack.auth.idpsyncpolicy.IdpSyncPolicyManager;
 import org.apache.cloudstack.framework.config.ConfigKey;
 import org.apache.cloudstack.saml.SAML2AuthManager;
 import org.apache.cloudstack.saml.SAMLPluginConstants;
@@ -78,7 +79,6 @@ import com.cloud.domain.Domain;
 import com.cloud.user.AccountService;
```

```
 import com.cloud.user.DomainManager;
 import com.cloud.user.UserAccountVO;
-import com.cloud.user.dao.UserAccountDao;
 import com.cloud.utils.HttpUtils;

 @RunWith(MockitoJUnitRunner.class)
@@ -97,7 +97,7 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
     AccountService accountService;

     @Mock
-    UserAccountDao userAccountDao;
+    IdpSyncPolicyManager idpSyncPolicyManager;

     @Mock
     Domain domain;
@@ -164,9 +164,9 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
         domainMgrField.setAccessible(true);
         domainMgrField.set(cmd, domainMgr);

-        Field userAccountDaoField = SAML2LoginAPIAuthenticatorCmd.class.
    ↪getDeclaredField("userAccountDao");
-        userAccountDaoField.setAccessible(true);
-        userAccountDaoField.set(cmd, userAccountDao);
+        Field idpSyncPolicyMgrField = SAML2LoginAPIAuthenticatorCmd.class.
    ↪getDeclaredField("idpSyncPolicyManager");
+        idpSyncPolicyMgrField.setAccessible(true);
+        idpSyncPolicyMgrField.set(cmd, idpSyncPolicyManager);

         KeyPair kp = CertUtils.generateRandomKeyPair(4096);
         X509Certificate cert = SAMLUtils.generateRandomX509Certificate(kp);
@@ -185,7 +185,7 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
         Mockito.lenient().when(domainMgr.getDomain(Mockito.anyString())).thenReturn
    ↪(domain);
         UserAccountVO user = new UserAccountVO();
         user.setId(1000L);
-        Mockito.lenient().when(userAccountDao.getUserAccount(Mockito.anyString(),
    ↪Mockito.anyLong())).thenReturn(user);
+        Mockito.lenient().when(idpSyncPolicyManager.login(Mockito.any(Map.class),
    ↪Mockito.anyString())).thenReturn(user);
```

```
         Mockito.lenient().when(apiServer.verifyUser(nullable(Long.class))).
    ↪thenReturn(false);
         Mockito.when(samlAuthManager.getSPMetadata()).thenReturn(providerMetadata);
         Mockito.when(samlAuthManager.getIdPMetadata(nullable(String.class))).
    ↪thenReturn(providerMetadata);
@@ -206,7 +206,7 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
             failing = false;
         }
         assertFalse("authentication should not have succeeded", failing);
-        Mockito.verify(userAccountDao, Mockito.times(0)).getUserAccount(Mockito.
    ↪anyString(), Mockito.anyLong());
+        Mockito.verify(idpSyncPolicyManager, Mockito.times(0)).login(Mockito.any(
    ↪Map.class), Mockito.anyString());
         Mockito.verify(apiServer, Mockito.times(0)).verifyUser(Mockito.anyLong());
     }

@@ -282,7 +282,7 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
         overrideDefaultConfigValue(SAML2AuthManager.SAMLCheckSignature, "_value",
    ↪false);
         SAML2LoginAPIAuthenticatorCmd cmd = new SAML2LoginAPIAuthenticatorCmd();
         try {
-            cmd.checkAndFailOnMissingSAMLSignature(null);
+            cmd.validateSignature(null, null, null, null);
         } catch(Exception e) {
             Assert.fail("This shouldn't throw any exception");
         }
@@ -292,7 +292,7 @@ public class SAML2LoginAPIAuthenticatorCmdTest {
     public void testFailOnSAMLSignatureCheckWhenTrue() throws NoSuchFieldException,
    ↪ IllegalAccessException {
         overrideDefaultConfigValue(SAML2AuthManager.SAMLCheckSignature, "_value",
    ↪true);
         SAML2LoginAPIAuthenticatorCmd cmd = new SAML2LoginAPIAuthenticatorCmd();
-        cmd.checkAndFailOnMissingSAMLSignature(null);
+        cmd.validateSignature(null, null, null, null);
     }

     private UserAccountVO
    ↪configureTestWhenFailToAuthenticateThrowExceptionOrRedirectToUrl(String entity,
    ↪ String configurationValue, Boolean isUserAuthorized)
```

A.38 SERVER/SRC/MAIN/JAVA/COM/CLOUD/USER/ ACCOUNTMANAGERIMPL.JAVA

```
diff --git a/server/src/main/java/com/cloud/user/AccountManagerImpl.java b/server/
    ↪src/main/java/com/cloud/user/AccountManagerImpl.java
index 39e8518f76..5116b66cd1 100644
--- a/server/src/main/java/com/cloud/user/AccountManagerImpl.java
+++ b/server/src/main/java/com/cloud/user/AccountManagerImpl.java
@@ -1224,7 +1224,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
                 accountCmd.getLastName(), accountCmd.getEmail(), accountCmd.
    ↪getTimeZone(), accountCmd.getAccountName(),
                 accountCmd.getAccountType(), accountCmd.getRoleId(), accountCmd.
    ↪getDomainId(),
                 accountCmd.getNetworkDomain(), accountCmd.getDetails(), accountCmd.
    ↪getAccountUUID(),
-                accountCmd.getUserUUID(), User.Source.UNKNOWN);
+                accountCmd.getUserUUID(), null, User.Source.UNKNOWN);
     }

     // ////////////////////////////////////////////////
@@ -1239,7 +1239,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
                                         final String lastName, final String email,
    ↪ final String timezone,
                                         String accountName, final Account.Type
    ↪accountType, final Long roleId, Long domainId,
                                         final String networkDomain, final Map<
    ↪String, String> details,
-                                        String accountUUID, final String userUUID,
    ↪ final User.Source source) {
+                                        String accountUUID, final String userUUID,
    ↪ String externalEntity, final User.Source source) {

         if (accountName == null) {
             accountName = userName;
@@ -1298,7 +1298,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
                 checkRoleEscalation(getCurrentCallingAccount(), account);
```

```
                // create the first user for the account
-               UserVO user = createUser(accountId, userName, password, firstName,
    ↪lastName, email, timezone, userUUID, source);
+               UserVO user = createUser(accountId, userName, password, firstName,
    ↪lastName, email, timezone, userUUID, externalEntity, source);

                if (accountType == Account.Type.RESOURCE_DOMAIN_ADMIN) {
                    // set registration token
@@ -1417,7 +1417,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
     @Override
     @ActionEvent(eventType = EventTypes.EVENT_USER_CREATE, eventDescription = "
    ↪creating User")
     public UserVO createUser(String userName, String password, String firstName,
    ↪String lastName, String email, String timeZone, String accountName, Long
    ↪domainId, String userUUID,
-            User.Source source) {
+                             String externalEntity, User.Source source) {
         // default domain to ROOT if not specified
         if (domainId == null) {
             domainId = Domain.ROOT_DOMAIN;
@@ -1445,7 +1445,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
             throw new CloudRuntimeException("The user " + userName + " already
    ↪exists in domain " + domainId);
         }
         UserVO user = null;
-        user = createUser(account.getId(), userName, password, firstName, lastName,
    ↪ email, timeZone, userUUID, source);
+        user = createUser(account.getId(), userName, password, firstName, lastName,
    ↪ email, timeZone, userUUID, externalEntity, source);
         return user;
     }

@@ -1453,7 +1453,7 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
     @ActionEvent(eventType = EventTypes.EVENT_USER_CREATE, eventDescription = "
    ↪creating User")
```

```
     public UserVO createUser(String userName, String password, String firstName,
    ↪String lastName, String email, String timeZone, String accountName, Long
    ↪domainId, String userUUID) {

-        return createUser(userName, password, firstName, lastName, email, timeZone,
    ↪ accountName, domainId, userUUID, User.Source.UNKNOWN);
+        return createUser(userName, password, firstName, lastName, email, timeZone,
    ↪ accountName, domainId, userUUID, null, User.Source.UNKNOWN);
     }

     @Override
@@ -2509,7 +2509,8 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
         });
     }

-    protected UserVO createUser(long accountId, String userName, String password,
    ↪String firstName, String lastName, String email, String timezone, String
    ↪userUUID, User.Source source) {
+    protected UserVO createUser(long accountId, String userName, String password,
    ↪String firstName, String lastName, String email, String timezone, String
    ↪userUUID,
+                                String externalEntity, User.Source source) {
         if (logger.isDebugEnabled()) {
             logger.debug("Creating user: " + userName + ", accountId: " + accountId
    ↪ + " timezone:" + timezone);
         }
@@ -2531,7 +2532,9 @@ public class AccountManagerImpl extends ManagerBase implements
    ↪ AccountManager, M
             userUUID = UUID.randomUUID().toString();
         }

-        UserVO user = _userDao.persist(new UserVO(accountId, userName,
    ↪encodedPassword, firstName, lastName, email, timezone, userUUID, source));
+        UserVO user = new UserVO(accountId, userName, encodedPassword, firstName,
    ↪lastName, email, timezone, userUUID, source);
+        user.setExternalEntity(externalEntity);
+        _userDao.persist(user);
         CallContext.current().putContextParameter(User.class, user.getUuid());
```

```
        return user;
    }
```

A.39 SERVER/SRC/MAIN/JAVA/COM/CLOUD/USER/PASSWORDPOLICYIMPL.JAVA

```
diff --git a/server/src/main/java/com/cloud/user/PasswordPolicyImpl.java b/server/
   ↪src/main/java/com/cloud/user/PasswordPolicyImpl.java
index daa57a4a50..1ad37bf00b 100644
--- a/server/src/main/java/com/cloud/user/PasswordPolicyImpl.java
+++ b/server/src/main/java/com/cloud/user/PasswordPolicyImpl.java
@@ -30,7 +30,7 @@ public class PasswordPolicyImpl implements PasswordPolicy,
   ↪Configurable {
     public void verifyIfPasswordCompliesWithPasswordPolicies(String password,
   ↪String username, Long domainId) {
         if (StringUtils.isEmpty(password)) {
             logger.warn(String.format("User [%s] has an empty password, skipping
   ↪password policy checks. " +
-                    "If this is not a LDAP user, there is something wrong.",
   ↪username));
+                    "If this is not a LDAP or IdP user, there is something wrong.",
   ↪ username));
             return;
         }
```

A.40 SERVER/SRC/MAIN/JAVA/ORG/APACHE/CLOUDSTACK/AUTH/IDPSYNCPOLICY/IDPSYNCPOLICYMANAGERIMPL.JAVA

```
// Licensed to the Apache Software Foundation (ASF) under one
// or more contributor license agreements.  See the NOTICE file
// distributed with this work for additional information
// regarding copyright ownership.  The ASF licenses this file
// to you under the Apache License, Version 2.0 (the
// "License"); you may not use this file except in compliance
// with the License.  You may obtain a copy of the License at
//
//   http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing,
```

```
// software distributed under the License is distributed on an
// "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, either express or implied.  See the License for the
// specific language governing permissions and limitations
// under the License.

package org.apache.cloudstack.auth.idpsyncpolicy;

import com.cloud.api.ApiDBUtils;
import com.cloud.domain.Domain;
import com.cloud.domain.dao.DomainDao;
import com.cloud.event.ActionEvent;
import com.cloud.event.EventTypes;
import com.cloud.exception.InvalidParameterValueException;
import com.cloud.exception.UnsupportedServiceException;
import com.cloud.user.Account;
import com.cloud.user.AccountManager;
import com.cloud.user.User;
import com.cloud.user.UserAccount;
import com.cloud.user.dao.AccountDao;
import com.cloud.user.dao.UserAccountDao;
import com.cloud.user.dao.UserDao;
import com.cloud.utils.StringUtils;
import com.cloud.utils.component.ManagerBase;
import com.cloud.utils.exception.CloudRuntimeException;
import com.google.gson.Gson;
import com.google.gson.JsonSyntaxException;
import org.apache.cloudstack.acl.Role;
import org.apache.cloudstack.acl.RoleType;
import org.apache.cloudstack.acl.dao.RoleDao;
import org.apache.cloudstack.api.ApiErrorCode;
import org.apache.cloudstack.api.ResponseGenerator;
import org.apache.cloudstack.api.ServerApiException;
import org.apache.cloudstack.api.command.admin.account.UpdateAccountCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.CreateIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.ListIdpSyncPoliciesCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.RemoveIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.UpdateIdpSyncPolicyCmd;
import org.apache.cloudstack.api.command.admin.idpsyncpolicy.
```

```
    ↪ValidateIdpSyncPolicyMappingCmd;
import org.apache.cloudstack.api.command.admin.user.UpdateUserCmd;
import org.apache.cloudstack.api.response.IdpSyncPolicyMappingResponse;
import org.apache.cloudstack.auth.idpsyncpolicy.dao.IdpSyncPolicyDao;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.AccountMap;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.AttributeMap;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.DomainMap;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.RoleMap;
import org.apache.cloudstack.auth.idpsyncpolicy.mapping.UserMap;
import org.apache.cloudstack.context.CallContext;
import org.apache.cloudstack.framework.config.ConfigKey;
import org.apache.cloudstack.framework.config.Configurable;
import org.apache.cloudstack.framework.config.dao.ConfigurationDao;
import org.apache.cloudstack.utils.jsinterpreter.JsInterpreter;
import org.apache.cloudstack.utils.reflectiontostringbuilderutils.
    ↪ReflectionToStringBuilderUtils;
import org.bouncycastle.util.encoders.Base64;
import org.springframework.stereotype.Component;

import javax.inject.Inject;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Consumer;

@Component
public class IdpSyncPolicyManagerImpl extends ManagerBase implements
    ↪IdpSyncPolicyManager, Configurable {

    @Inject
    IdpSyncPolicyDao idpSyncPolicyDao;
    @Inject
```

```
    UserAccountDao userAccountDao;
    @Inject
    UserDao userDao;
    @Inject
    DomainDao domainDao;
    @Inject
    AccountDao accountDao;
    @Inject
    RoleDao roleDao;
    @Inject
    AccountManager accountManager;
    @Inject
    ConfigurationDao configurationDao;
    @Inject
    ResponseGenerator responseGenerator;

    private final Gson gson = new Gson();
    private static final String DEFAULT_MAPPING = "r = {\"user\": { \"username\":
↪idp.%s }, \"legacy\": true }";
    private static final String AUTOCREATED_DESCRIPTION = "Auto created default
↪policy.";

    @Override
    @ActionEvent(eventType = EventTypes.EVENT_IDPSYNCPOLICY_CREATE, eventDescription
↪ = "creating IdP Sync Policy")
    public IdpSyncPolicy createIdpSyncPolicy(CreateIdpSyncPolicyCmd cmd) {
        CallContext.current().setEventDetails("IdP: " + cmd.getIdpId());
        String idpId = cmd.getIdpId();

        if (idpSyncPolicyDao.findActiveByIdpId(idpId) != null) {
            throw new InvalidParameterValueException(String.format("IdP [%s] already
↪ has an active Sync Policy.", idpId));
        }

        IdpSyncPolicyVO vo = new IdpSyncPolicyVO(idpId, cmd.getUserAccountOperation
↪(), cmd.getMapping(), cmd.getDescription());
        IdpSyncPolicyVO persisted = idpSyncPolicyDao.persist(vo);
        CallContext.current().setEventResourceId(vo.getId());
        return persisted;
```

```
}

private IdpSyncPolicyVO getActiveIdpSyncPolicyOrThrow(Long id) {
    IdpSyncPolicyVO vo = idpSyncPolicyDao.findById(id);
    if (vo == null) {
        logger.error("Active IdP Sync Policy with ID [{}] was not found.", id);
        throw new ServerApiException(ApiErrorCode.RESOURCE_UNAVAILABLE_ERROR, "
↪Active IdP Sync Policy not found.");
    }
    return vo;
}

@Override
@ActionEvent(eventType = EventTypes.EVENT_IDPSYNCPOLICY_UPDATE, eventDescription
↪ = "updating IdP Sync Policy")
public IdpSyncPolicy updateIdpSyncPolicy(UpdateIdpSyncPolicyCmd cmd) {
    Long id = cmd.getId();
    String description = cmd.getDescription();
    String mapping = cmd.getMapping();

    IdpSyncPolicyVO vo = getActiveIdpSyncPolicyOrThrow(id);
    if (description == null && mapping == null) {
        logger.debug("No changes requested. Not updating IdP Sync Policy [{}].",
↪ vo);
        return vo;
    }
    CallContext.current().setEventDetails("IdP: " + vo.getIdpId());

    if (description != null) {
        vo.setDescription(description);
    }
    if (mapping != null) {
        vo.setMapping(mapping);
    }
    vo.incrUpdateCount();
    vo.setUpdated(new Date());
    return idpSyncPolicyDao.persist(vo);
}
```

```
@Override
@ActionEvent(eventType = EventTypes.EVENT_IDPSYNCPOLICY_REMOVE, eventDescription
↪ = "removing IdP Sync Policy")
public IdpSyncPolicy removeIdpSyncPolicy(RemoveIdpSyncPolicyCmd cmd) {
    Long id = cmd.getId();
    String removalReason = cmd.getRemovalReason();

    IdpSyncPolicyVO vo = getActiveIdpSyncPolicyOrThrow(id);
    CallContext.current().setEventDetails("IdP: " + vo.getIdpId());

    vo.setRemovalReason(removalReason);
    vo.setRemoved(new Date());
    return idpSyncPolicyDao.persist(vo);
}

@Override
public List<? extends IdpSyncPolicy> listIdpSyncPolicies(ListIdpSyncPoliciesCmd
↪cmd) {
    Long id = cmd.getId();
    String idpId = cmd.getIdpId();
    boolean showRemoved = cmd.showRemoved();
    Long startIndex = cmd.getStartIndex();
    Long pageSize = cmd.getPageSizeVal();
    String keyword = cmd.getKeyword();

    logger.debug("Searching for IdP Sync Policies that match params [id: {},
↪idpId: {}, showRemoved: {}, startIndex: {}, pageSize: {}, keyword: {}].",
            id, idpId, showRemoved, startIndex, pageSize, keyword);
    return idpSyncPolicyDao.listBy(id, idpId, showRemoved, startIndex, pageSize,
↪ keyword);
}

@Override
public IdpSyncPolicyMappingResponse validateIdpSyncPolicyMapping(
↪ValidateIdpSyncPolicyMappingCmd cmd) {
    String mapping = cmd.getMapping();
    String idpJson = cmd.getIdpJson();
    IdpSyncPolicy.Operation userAccountOperation = cmd.getUserAccountOperation()
↪;
```

```
        AttributeMap mapped = mapAttributes(idpJson, mapping);

        IdpSyncPolicyMappingResponse response = new IdpSyncPolicyMappingResponse();
        response.setMappedAttributes(mapped);
        try {
            UserAccount userAccount = runPolicy(mapped, userAccountOperation, null);
            response.setUserResponse(responseGenerator.createUserResponse(
↪userAccount));
        } catch (CloudRuntimeException e) {
            if (!(e instanceof UnsupportedServiceException)) {
                response.setSuccess(Boolean.FALSE);
            }
            response.setDisplayText(e.getMessage());
        }
        return response;
    }

    protected AttributeMap mapAttributes(String idpAttributes, String mappingScript)
↪ throws ServerApiException {
        Object result;
        try (JsInterpreter jsInterpreter = new JsInterpreter(
↪IdpSyncPolicyMappingTimeout.value())) {
            jsInterpreter.injectVariable("idp", idpAttributes);
            result = jsInterpreter.executeScript(mappingScript);
        } catch (IOException | CloudRuntimeException e) {
            logger.error("The interpreter could not map attributes.");
            throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Error
↪executing script.");
        }

        try {
            String json = gson.toJson(result);
            logger.trace("Attributes resolved from script: [{}]", json);
            AttributeMap map = gson.fromJson(json, AttributeMap.class);
            if (logger.isDebugEnabled()) {
                logger.debug("Mapped attributes: [{}]", gson.toJson(map));
            }
            return map;
        }
```

```
        catch (JsonSyntaxException e) {
            logger.error("JSON Error during IdP attribute mapping:", e);
            throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Error
↪executing script.");
        }
    }

    @Override
    public UserAccount login(Map<String, Object> attributes, String idpId) {
        IdpSyncPolicy policy = idpSyncPolicyDao.findActiveByIdpId(idpId);
        if (policy == null) {
            logger.error("Could not find IdP Sync Policy for [{}].", idpId);
            return null;
        }

        String idpAttributes = gson.toJson(attributes);
        logger.debug("Attributes received from IdP [{}] are [{}].", idpId,
↪idpAttributes);
        AttributeMap map = mapAttributes(idpAttributes, policy.getMapping());

        try {
            UserAccount userAccount = runPolicy(map, policy.getOperation(), idpId);
            logger.info("User [{}] returned by IdP Sync Policy.", reflectUserAccount
↪(userAccount));
            return userAccount;
        } catch (CloudRuntimeException e) {
            logger.error("Could not login due to: ", e);
            return null;
        }
    }

    @Override
    @ActionEvent(eventType = EventTypes.EVENT_IDPSYNCPOLICY_CREATE, eventDescription
↪ = "creating Default IdP Sync Policy")
    public void createDefaultPolicy(String idpId, String username) {
        if (!IdpSyncPolicyAutoCreateDefaultPolicy.value()) {
            CallContext.current().setEventType("");
            return;
        }
```

```
        if (idpSyncPolicyDao.findAnyByIdpId(idpId) != null) {
            CallContext.current().setEventType("");
            logger.trace("Not creating default policy since a policy has already
↪been created.");
            return;
        }
        CreateIdpSyncPolicyCmd cmd = new CreateIdpSyncPolicyCmd();
        cmd.setIdpId(idpId);
        cmd.setMapping(String.format(DEFAULT_MAPPING, username));
        cmd.setDescription(AUTOCREATED_DESCRIPTION);
        try {
            IdpSyncPolicy policy = createIdpSyncPolicy(cmd);
            logger.info("Auto created default policy [{}].", policy);
        } catch (CloudRuntimeException e) {
            logger.warn("Could not auto create default policy for [{}] due to: {}",
↪idpId, e.getMessage());
        }
    }

    private void throwIfNoSamlPermissionOrNotEnabled(UserAccount userAccount, String
↪ idpId) throws ServerApiException {
        if (!userAccount.getAccountState().equals(Account.State.ENABLED.toString()))
↪ {
            throw new ServerApiException(ApiErrorCode.UNAUTHORIZED, String.format("
↪UserAccount [%s] is in state [%s].",
                    reflectUserAccount(userAccount), userAccount.getAccountState()))
↪;
        }
        String externalEntity = userAccount.getExternalEntity();
        if (!userAccount.getSource().equals(User.Source.SAML2) || externalEntity ==
↪null || !externalEntity.equalsIgnoreCase(idpId)) {
            throw new ServerApiException(ApiErrorCode.UNAUTHORIZED, String.format("
↪User [%s] does not have permission to SAML login with IdP [%s].",
                    reflectUserAccount(userAccount), idpId));
        }
    }

    /**
     * Allows for policies with only "username" so we can keep backwards compatible
```

```
↪with original SAML behavior. This is deprecated and should be removed when
↪possible.
  * @return The first of any users who fit the username and have SAML2 source.
  */
 private UserAccount bypass(String username, String idpId) {
     List<? extends UserAccount> possibleUserAccounts = userAccountDao.
↪getAllUsersByNameAndEntity(username, idpId);
     if (possibleUserAccounts == null || possibleUserAccounts.isEmpty()) {
         return null;
     }
     for (UserAccount possibleUserAccount : possibleUserAccounts) {
         if (possibleUserAccount.getAccountState().equals(Account.State.ENABLED.
↪toString())) {
             return possibleUserAccount;
         }
     }
     return null;
 }

 /**
  * Searches for existing UserAccount. If not found and operation allows, create
↪a new one. If found and operation allows, check if attributes are different and
↪ update.
  * @param idpId ID of IdP that sent the attributes. If null, it means this is a
↪trial run and should not persist any changes to database.
  * @throws CloudRuntimeException Explanation of policy execution failure.
  */
 private UserAccount runPolicy(AttributeMap map, IdpSyncPolicy.Operation
↪operation, String idpId) throws CloudRuntimeException {
     if (map.shouldBypass()) {
         logger.warn("Bypassing usual flow due to policy mapping containing a
↪single \"username\" attribute as well as 'legacy' being true.");
         return bypass(map.getUser().getUsername(), idpId);
     }

     if (map.getUser() == null) {
         throw new CloudRuntimeException("Can't login since no user attributes
↪were received.");
     }
```

```
        if (map.getUser().getPassword() != null) {
            logger.warn("Password field found, but it is not expected and will not
↪be used.");
        }

        Domain domain = findDomain(map.getDomain());
        User user = findUser(map.getUser(), domain);
        Account account = findAccount(map.getAccount(), domain);
        if (domain == null && account != null) {
            domain = domainDao.findById(account.getDomainId());
        }

        if (user == null) {
            if (!operationCanCreate(operation)) {
                throw new CloudRuntimeException(String.format("User not found and
↪operation [%s] doesn't allow creation.", operation));
            }
            return validateMappingAndCreateUserAccount(map, domain, account, idpId);
        }

        UserAccount userAccount = userAccountDao.findById(user.getId());
        throwIfNoSamlPermissionOrNotEnabled(userAccount, idpId);

        if (!operationCanUpdate(operation)) {
            logger.debug("Returning user [{}]. Not checking for different attributes
↪ since operation [{}] doesn't allow updating.", user, operation);
            return userAccount;
        }
        return validateMappingAndUpdateUserAccountIfNeeded(map, userAccount, account,
↪ idpId);
    }

    private UserAccount validateMappingAndCreateUserAccount(AttributeMap map, Domain
↪ domain, Account account, String idpId) throws CloudRuntimeException {
        if (domain == null) {
            throw new CloudRuntimeException("User and domain not found, so we can't
↪create new UserAccount.");
        }
        boolean simulate = idpId == null;
```

```
        String userUuid = map.getUser().getUuid();
        String username = map.getUser().getUsername();
        String firstName = map.getUser().getFirstname();
        String lastName = map.getUser().getLastname();
        String email = map.getUser().getEmail();
        String timezone = map.getUser().getTimezone();
        User.Source source = User.Source.SAML2;

        if (account != null) {
            if (simulate) {
                throw new UnsupportedServiceException(String.format("Would try to
↪create user in account [%s] of domain [%s].", account, domain.getPath()));
            }
            User user = accountManager.createUser(username, generatePassword(),
↪firstName, lastName, email, timezone, account.getAccountName(), domain.getId(),
↪ userUuid, idpId, source);
            return userAccountDao.findById(user.getId());
        }

        if (map.getAccount() == null) {
            throw new CloudRuntimeException("User not found and no account
↪information was given, so we can't create new account.");
        }

        Role role = findRole(map.getAccount().getRole());
        if (role == null) {
            throw new CloudRuntimeException("User, account and role not found, so we
↪ can't create new account.");
        }
        String accountUuid = map.getAccount().getUuid();
        String accountName = map.getAccount().getAccountName();
        Account.Type accountType = role.getRoleType().getAccountType();
        String networkDomain = map.getAccount().getNetworkDomain();
        Map<String, String> details = map.getAccount().getDetails();

        if (simulate) {
            map.getUser().hidePassword();
            throw new UnsupportedServiceException(String.format("Would try to create
```

```
↪ UserAccount in domain [%s].", domain.getPath()));
    }

    return accountManager.createUserAccount(username, generatePassword(),
↪firstName, lastName, email, timezone, accountName, accountType, role.getId(),
↪domain.getId(),
            networkDomain, details, accountUuid, userUuid, idpId, source);
 }


 private UserAccount validateMappingAndUpdateUserAccountIfNeeded(AttributeMap map,
↪ UserAccount userAccount, Account account, String idpId) throws
↪CloudRuntimeException {
    boolean simulate = idpId == null;
    if (account != null) {
        if (validateAndUpdateAccountIfNeeded(map.getAccount(), account, simulate
↪)) {
            userAccount = userAccountDao.findById(userAccount.getId());
        }
    }

    String reflectedUser = reflectUserAccount(userAccount);
    logger.trace("Verifying user [{}] attributes.", reflectedUser);
    UpdateUserCmd cmd = new UpdateUserCmd();
    boolean hasUpdate = setStringValueIfNotEqual(cmd::setEmail, map.getUser().
↪getEmail(), userAccount.getEmail());
    hasUpdate = setStringValueIfNotEqual(cmd::setFirstname, map.getUser().
↪getFirstname(), userAccount.getFirstname()) || hasUpdate;
    hasUpdate = setStringValueIfNotEqual(cmd::setLastname, map.getUser().
↪getLastname(), userAccount.getLastname()) || hasUpdate;
    hasUpdate = setStringValueIfNotEqual(cmd::setUsername, map.getUser().
↪getUsername(), userAccount.getUsername()) || hasUpdate;
    hasUpdate = setStringValueIfNotEqual(cmd::setTimezone, map.getUser().
↪getTimezone(), userAccount.getTimezone()) || hasUpdate;

    if (!hasUpdate) {
        logger.debug("Returning user [{}] unchanged, since attributes were equal.
↪", reflectedUser);
        return userAccount;
```

```
        }

        String reflectedCmd = ReflectionToStringBuilderUtils.
↪reflectOnlySelectedFields(cmd, "email", "firstname", "lastname", "username", "
↪timezone");
        if (simulate) {
            throw new UnsupportedServiceException(String.format("Would try to update
↪ user [%s] with parameters [%s].", reflectedUser,reflectedCmd));
        }

        logger.debug("Updating user [{}] with new params [{}].", reflectedUser,
↪reflectedCmd);
        cmd.setId(userAccount.getId());
        return accountManager.updateUser(cmd);
    }

    private boolean validateAndUpdateAccountIfNeeded (AccountMap map, Account
↪account, boolean simulate) {
        logger.trace("Verifying account [{}] attributes.", account);
        UpdateAccountCmd cmd = new UpdateAccountCmd();
        boolean hasUpdate = setStringValueIfNotEqual(cmd::setNewName, map.
↪getAccountName(), account.getAccountName());
        hasUpdate = setStringValueIfNotEqual(cmd::setNetworkDomain, map.
↪getNetworkDomain(), account.getNetworkDomain()) || hasUpdate;

        if (map.getDetails() != null) {
            Map<String, String> details = map.getDetails();
            Map<String, String> accountDetails = ApiDBUtils.getAccountDetails(
↪account.getId());
            if (details != null && !details.equals(accountDetails)) {
                hasUpdate = true;
                cmd.setDetails(detailsToCmdParam(details));
            }
        }

        Role role = findRole(map.getRole());
        if (role != null && account.getRoleId() != role.getId()) {
            hasUpdate = true;
            cmd.setRoleId(role.getId());
```

```
        }

        if (!hasUpdate) {
            logger.debug("Not updating [{}], since attributes were equal.", account)
↪;
            return false;
        }

        if (simulate) {
            throw new UnsupportedServiceException(String.format("Would try to update
↪ [%s] with parameters [%s].", account,
                    ReflectionToStringBuilderUtils.reflectOnlySelectedFields(cmd, "
↪accountName", "networkDomain", "details", "roleId")));
        }
        accountManager.updateAccount(cmd);
        return true;
    }

    private Map detailsToCmdParam (Map<String, String> details) {
        Map<Integer, Map<String,String>> apiParam = new HashMap<>();
        apiParam.put(0, details);
        return apiParam;
    }

    protected boolean setStringValueIfNotEqual(Consumer<String> setter, String
↪newValue, String oldValue){
        if (StringUtils.isNotBlank(newValue) && !newValue.equalsIgnoreCase(oldValue)
↪) {
            setter.accept(newValue);
            return true;
        }
        return false;
    }

    private Domain findDomain(DomainMap map) {
        if (map == null) {
            logger.trace("No domain attributes received.");
            return null;
        }
```

```
        String uuid = map.getUuid();
        if (uuid != null) {
            logger.trace("Searching for domain with UUID [{}].", uuid);
            return domainDao.findByUuid(uuid);
        }
        String path = map.getPath();
        if (path == null) {
            return null;
        }
        if (!path.startsWith("/")) {
            path = "/" + path;
        }
        if (!path.endsWith("/")) {
            path = path + "/";
        }
        logger.trace("Searching for domain [{}].", path);
        return domainDao.findDomainByPath(path);
    }

    private boolean operationCanCreate(IdpSyncPolicy.Operation operation) {
        return operation == IdpSyncPolicy.Operation.CREATE || operation ==
↪IdpSyncPolicy.Operation.CREATEANDUPDATE;
    }

    private boolean operationCanUpdate(IdpSyncPolicy.Operation operation) {
        return operation == IdpSyncPolicy.Operation.UPDATE || operation ==
↪IdpSyncPolicy.Operation.CREATEANDUPDATE;
    }

    private User findUser(UserMap user, Domain domain) {
        String uuid = user.getUuid();
        if (uuid != null) {
            logger.trace("Searching for user with UUID [{}].", uuid);
            return userDao.findByUuid(uuid);
        }
        String username = user.getUsername();
        if (username == null || domain == null) {
            return null;
        }
```

```
        logger.trace("Searching for user with username [{}] and domain [{}].",
↪username, domain.getId());
        return userDao.getUserByName(username, domain.getId());
    }

    private Account findAccount(AccountMap map, Domain domain) {
        if (map == null) {
            logger.trace("No account attributes received.");
            return null;
        }
        String uuid = map.getUuid();
        if (uuid != null) {
            logger.trace("Searching for account with UUID [{}].", uuid);
            return accountDao.findByUuid(uuid);
        }
        String accountName = map.getAccountName();
        if (accountName == null || domain == null) {
            return null;
        }
        logger.trace("Searching for account with name [{}] and domain [{}].",
↪accountName, domain.getId());
        return accountDao.findActiveAccount(accountName, domain.getId());
    }

    private Role findRole(RoleMap map) {
        if (map == null) {
            logger.trace("No role attributes received.");
            return null;
        }
        if (map.getUuid() != null) {
            logger.trace("Searching for role with UUID [{}].", map.getUuid());
            return roleDao.findByUuid(map.getUuid());
        }
        String name = map.getName();
        String typeStr = map.getType();
        if (name == null) {
            return null;
        }
        if (typeStr == null) {
```

```
            return findRoleByOnlyName(name);
        }
        RoleType type;
        try {
            type = RoleType.valueOf(typeStr);
        } catch (IllegalArgumentException e) {
            logger.error("Invalid Role Type [{}].", typeStr);
            return null;
        }
        logger.trace("Searching for role with name [{}] and type [{}].", map.getUuid
↪());
        return roleDao.findByNameAndType(name, type, true);
    }

    private Role findRoleByOnlyName (String name) {
        List<? extends Role> roles = roleDao.findAllByName(name, true);
        if (roles != null && roles.size() == 1) {
            return roles.get(0);
        }
        return null;
    }

    private String generatePassword() throws ServerApiException {
        try {
            final SecureRandom randomGen = SecureRandom.getInstance("SHA1PRNG");
            final byte[] bytes = new byte[20];
            randomGen.nextBytes(bytes);
            return new String(Base64.encode(bytes), StandardCharsets.UTF_8);
        } catch (NoSuchAlgorithmException e) {
            throw new ServerApiException(ApiErrorCode.INTERNAL_ERROR, "Failed to
↪generate random password.");
        }
    }

    private String reflectUserAccount(UserAccount userAccount) {
        return ReflectionToStringBuilderUtils.reflectOnlySelectedFields(userAccount,
↪ "id", "username", "accountId", "accountName", "domainId");
    }
```

```
    @Override
    public List<Class<?>> getCommands() {
        String samlEnabled = configurationDao.getValue("saml2.enabled");
        if (samlEnabled == null || samlEnabled.equals("false")) {
            logger.debug("Ignoring IdP Sync Policy commands since saml2.enabled is
    ↪not set.");
            return Collections.emptyList();
        }
        List<Class<?>> cmdList = new ArrayList<>();
        cmdList.add(CreateIdpSyncPolicyCmd.class);
        cmdList.add(UpdateIdpSyncPolicyCmd.class);
        cmdList.add(RemoveIdpSyncPolicyCmd.class);
        cmdList.add(ListIdpSyncPoliciesCmd.class);
        cmdList.add(ValidateIdpSyncPolicyMappingCmd.class);
        return cmdList;
    }

    @Override
    public String getConfigComponentName() {
        return IdpSyncPolicyManager.class.getSimpleName();
    }

    @Override
    public ConfigKey<?>[] getConfigKeys() {
        return new ConfigKey<?>[] {
                IdpSyncPolicyMappingTimeout, IdpSyncPolicyAutoCreateDefaultPolicy
        };
    }
}
```

A.41 SERVER/SRC/MAIN/RESOURCES/META-INF/CLOUDSTACK/CORE/ SPRING-SERVER-CORE-MANAGERS-CONTEXT.XML

```
diff --git a/server/src/main/resources/META-INF/cloudstack/core/spring-server-core-
    ↪managers-context.xml b/server/src/main/resources/META-INF/cloudstack/core/
    ↪spring-server-core-managers-context.xml
index 1bf921f625..3e194d8bc0 100644
--- a/server/src/main/resources/META-INF/cloudstack/core/spring-server-core-managers-
    ↪context.xml
+++ b/server/src/main/resources/META-INF/cloudstack/core/spring-server-core-managers-
```

```
   ↪context.xml
@@ -84,6 +84,8 @@

     <bean id="resourceCleanupServiceImpl" class="org.apache.cloudstack.resource.
   ↪ResourceCleanupServiceImpl" />

+    <bean id="idpSyncPolicyManagerImpl" class="org.apache.cloudstack.auth.
   ↪idpsyncpolicy.IdpSyncPolicyManagerImpl" />
+
     <!-- the new background poll manager -->
     <bean id="bgPollManager" class="org.apache.cloudstack.poll.
   ↪BackgroundPollManagerImpl">
     </bean>
```

A.42 SERVER/SRC/TEST/JAVA/COM/CLOUD/USER/ MOCKACCOUNTMANAGERIMPL.JAVA

```
diff --git a/server/src/test/java/com/cloud/user/MockAccountManagerImpl.java b/
   ↪server/src/test/java/com/cloud/user/MockAccountManagerImpl.java
index bd6632af1c..ffe213e2ed 100644
--- a/server/src/test/java/com/cloud/user/MockAccountManagerImpl.java
+++ b/server/src/test/java/com/cloud/user/MockAccountManagerImpl.java
@@ -372,13 +372,13 @@ public class MockAccountManagerImpl extends ManagerBase
   ↪implements Manager, Acco
                 cmd.getLastName(), cmd.getEmail(), cmd.getTimeZone(), cmd.
   ↪getAccountName(),
                 cmd.getAccountType(), cmd.getRoleId(), cmd.getDomainId(),
                 cmd.getNetworkDomain(), cmd.getDetails(), cmd.getAccountUUID(),
-                cmd.getUserUUID(), User.Source.UNKNOWN);
+                cmd.getUserUUID(), null, User.Source.UNKNOWN);
     }

     @Override
     public UserAccount createUserAccount(String userName, String password, String
   ↪firstName, String lastName, String email, String timezone, String accountName,
                                          Account.Type accountType, Long roleId,
   ↪Long domainId, String networkDomain, Map<String, String> details, String
   ↪accountUUID,
-                                         String userUUID, User.Source source) {
+                                         String userUUID, String externalEntity,
```

```
    ↪User.Source source) {
          // TODO Auto-generated method stub
          return null;
      }
@@ -392,7 +392,7 @@ public class MockAccountManagerImpl extends ManagerBase
    ↪implements Manager, Acco
      }

      @Override public User createUser(String userName, String password, String
    ↪firstName, String lastName, String email, String timeZone, String accountName,
    ↪Long domainId,
-                                       String userUUID, User.Source source) {
+                                       String userUUID, String externalEntity, User.
    ↪Source source) {
          // TODO Auto-generated method stub
          return null;
      }
```

A.43 TOOLS/APIDOC/ GEN_TOC.PY

```
diff --git a/tools/apidoc/gen_toc.py b/tools/apidoc/gen_toc.py
index 8d28749a63..f50c3003fe 100644
--- a/tools/apidoc/gen_toc.py
+++ b/tools/apidoc/gen_toc.py
@@ -122,6 +122,7 @@ known_categories = {
      'Event': 'Event',
      'login': 'Authentication',
      'logout': 'Authentication',
+     'IdpSyncPolic': 'Authentication',
      'saml': 'Authentication',
      'getSPMetadata': 'Authentication',
      'listIdps': 'Authentication',
```

APÊNDICE B – ARTIGO

Implementação de controle de acesso federado baseado em atributos para ambientes de nuvem Apache CloudStack

Gabriel Pordeus Santos[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
88.040-970 – Florianópolis – SC – Brasil

gabriel.pordeus@grad.ufsc.br

Abstract. *This paper aims to extend Apache CloudStack so it can create and update users according to attributes received from identity providers during the SAML Single Sign-On process. Current SAML authentication flow is diagrammed and a specification for the propposed changes is created. Existing functionalities are reused to allow for the attributes received from the IdP to be mapped to a pre-defined structure. Adaptations are made to maintain compatibility to the system previous behavior. The solution is implemented and tested, and future works are suggested.*

Resumo. *Este trabalho tem como objetivo estender o Apache CloudStack para que crie e atualize usuários de acordo com os atributos recebidos dos provedores de identidade durante o processo de Single Sign-On via SAML. O fluxo de autenticação SAML atual do Apache CloudStack é diagramado e é criada uma especificação das mudanças propostas. Funcionalidades existentes são reaproveitadas para possibilitar o mapeamento dos atributos recebidos do IdP para uma estrutura pré-definida. São realizadas adaptações para manter a compatibilidade com o comportamento prévio do sistema. A solução é implementada e testada e trabalhos futuros são sugeridos.*

1. Introdução

A gestão de identidade é essencial para garantir a segurança e a experiência do usuário em ambientes de nuvem, exigindo planejamento e desenvolvimento para atender às necessidades de negócios e às demandas de controle de acesso [Wilson and Hingnikar 2019]. A aplicação tradicional de credenciais de usuário e senha tem limitações e é inadequada para sistemas de rede modernos, sendo necessária a integração de soluções como *Single Sign-On* (SSO) e *Single Logout* e a implementação de federações de identidade [ITU-T 2009].

O Apache CloudStack (ACS) é uma plataforma de orquestração de nuvem de código aberto, parte da fundação Apache e aberta a contribuições da comunidade [ASF 2024a]. Embora ofereça suporte a SSO via SAML, sua implementação limita-se a utilizar um único atributo do provedor de identidade para identificar o usuário, sem considerar outros atributos que poderiam auxiliar no processo de autenticação e autorização.

Este trabalho propõe uma extensão ao Apache CloudStack, permitindo que ele identifique e utilize todos os atributos presentes na resposta SAML durante o processo de *login* para a identificação, criação e atualização dos usuários.

2. Gestão de Identidade

Uma identidade é um conjunto de atributos associados a uma entidade (seja um usuário, elemento de rede, aplicação de software, entre outros) em um contexto específico [Wilson and Hingnikar 2019, ITU-T 2009]. No escopo deste trabalho, consideramos este contexto como um sistema de informação.

Um provedor de identidade (*Identity Provider* – IdP) é a entidade que cria, mantém e gerencia informações confiáveis de identidade, oferecendo serviços relacionados a essa confiabilidade — como a autenticação de usuários e o fornecimento de asserções sobre eles [ITU-T 2009, Chadwick 2009].

Um provedor de serviço (*Service Provider* – SP) é a entidade que precisa conhecer o quão confiáveis são as asserções vinculadas às credenciais fornecidas por seus usuários, para que possa responder às suas requisições [Bertino and Takahashi 2011].

3. Apache CloudStack

Computação em nuvem é um modelo de consumo, por rede e sob demanda, de recursos computacionais (como redes, servidores, armazenamento, aplicações e serviços) que podem ser implantados e removidos com o mínimo de esforço e interação [Mell and Grance 2011].

Dentre os tipos de serviços prestados por ambientes de computação em nuvem, está a infraestrutura como serviço (*Infrastructure as a Service* — IaaS), que permite ao consumidor preparar e executar qualquer *software*, incluindo sistemas operacionais. O consumidor não controla a infraestrutura física diretamente, mas pode configurar certos aspectos de rede [Mell and Grance 2011].

O Apache CloudStack (ACS) é um *software* de código aberto que oferece a orquestração de máquinas virtuais, *Network as a Service*, gerência de contas e usuários, contabilidade de recursos e outros processos gerenciais necessários em um ambiente de nuvem [ASF 2024a].

No ACS, o processo de autenticação pode ser feito através da assinatura da requisição à API com um par de chaves ou através do estabelecimento de uma sessão. A sessão pode ser aberta com credenciais internas (nome de usuário, senha e domínio) ou através de IdPs externos (por SAML 2.0 ou OAuth2) [ASF 2024b].

No *login* via SAML, o ACS procura pelo valor de um atributo (cujo nome é previamente definido pelo administrador), que contém o nome de usuário que está sendo autenticado [ASF 2024b]. Também é necessário que o administrador dê permissão ao usuário (através de um *endpoint* específico, `authorizeSamlSso`) antes que ele consiga se logar por SAML. No caso de múltiplos usuários com SAML habilitado e com o mesmo nome, após o *login* pode-se trocar entre usuários através do *endpoint* `listAndSwitchSamlAccount`.

4. Política de Sincronização com IdP

Política de sincronização é o nome dado ao conjunto que reúne: um *script* de mapeamento dos dados recebidos; a URL do IdP que forneceu os dados; e o tipo de operação a ser realizada com estes dados.

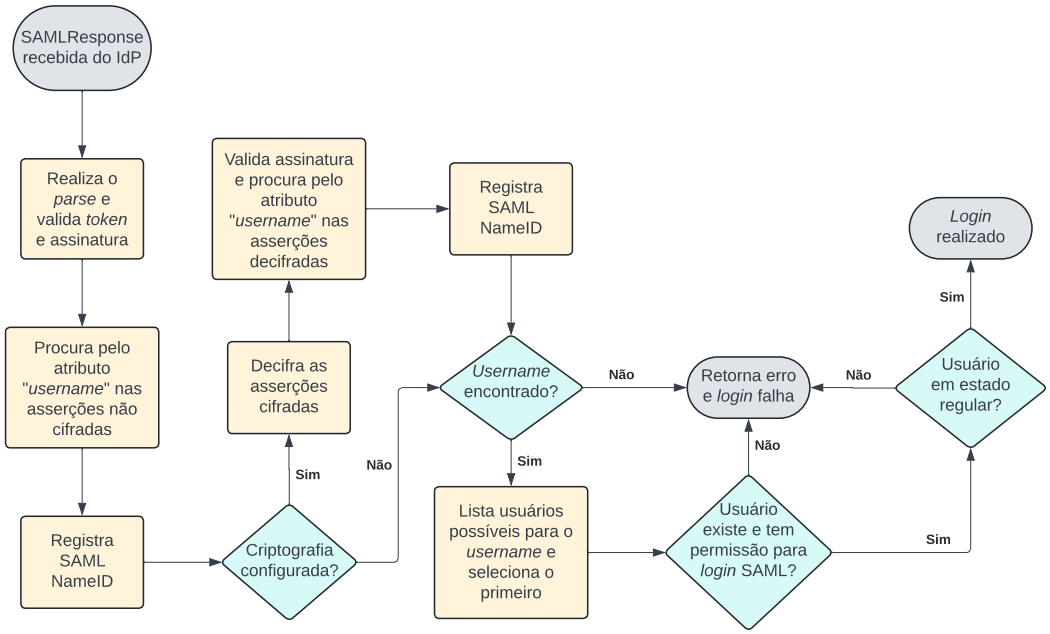

Figure 1. Fluxograma do *login* SAML presente no ACS.

A Figura 1 descreve o processo atualmente executado pelo ACS para validação da resposta recebida do IdP após o processo de autenticação do usuário. Ele procura pelo atributo com determinado nome (previamente configurado pelo administrador em ambos ACS e IdP) e realiza o *login* do primeiro usuário encontrado cujo *username* seja igual ao valor do atributo. Uma vez logado, o usuário pode trocar entre contas com mesmo *username* através de outro *endpoint* da API.

Em contraste, o fluxo proposto se encontra na Figura 2. Ao invés de procurar apenas o atributo contendo o *username*, todos os atributos serão repassados ao controlador da política de sincronização. A seção ”Agregação dos atributos SAML” é especifica ao tratamento do protocolo SAML, de acordo com o escopo do trabalho, mas a parte de mapeamento e execução das políticas de acordo com a operação é agnóstica e pode ser aplicada a atributos recebidos por quaisquer protocolos, sendo necessário estender as respectivas implementações no ACS para que realizem esta agregação.

Os atributos são injetados no interpretador JavaScript, que executa o *script* e retorna um objeto que representa um usuário no contexto do ACS. Se o usuário equivalente não existir no banco de dados de metadados do ACS e a política permitir, ele é criado. Se houver informações conflitantes com o usuário existente e a política permitir, ele é atualizado.

Para manutenção da compatibilidade com o comportamento atual da plataforma, será adicionada a possibilidade de criação automática de uma política padrão (que mantenha o mesmo fluxo atual) ao encontrar um novo IdP.

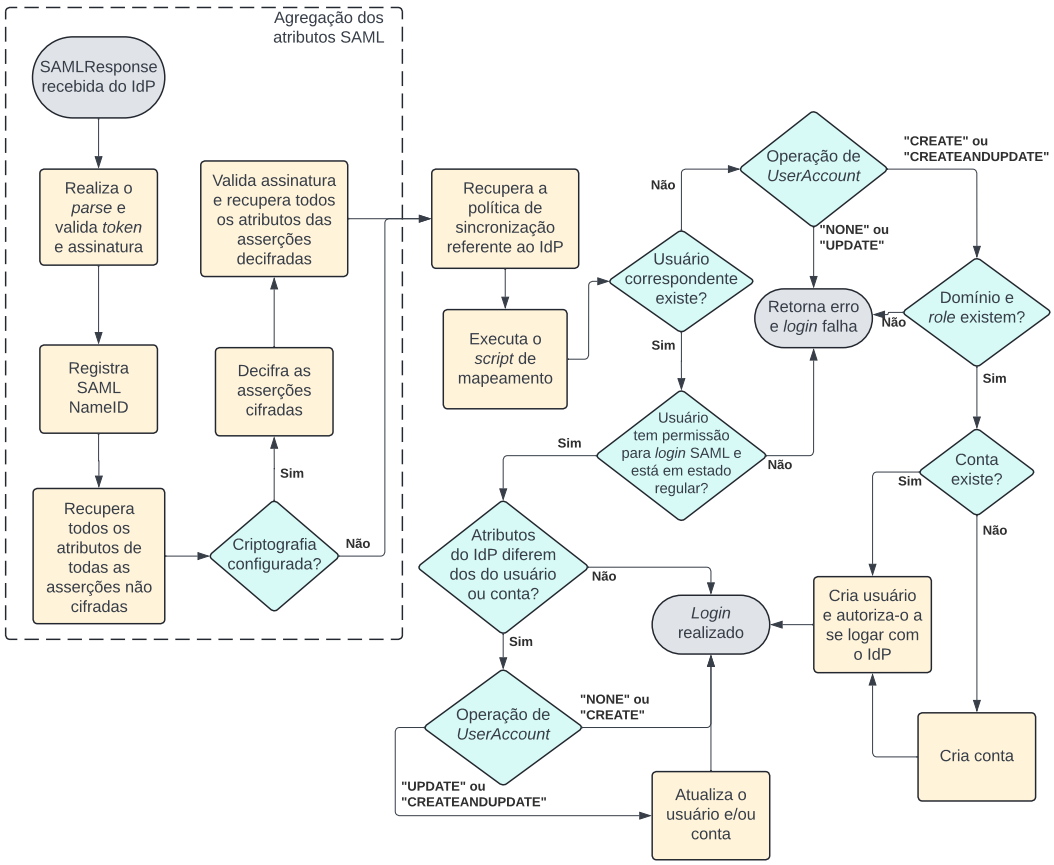

Figure 2. Fluxograma proposto para o *login* SAML do ACS.

4.1. Operação

Nesta primeira versão, o único tipo de operação presente será o de *UserAccounts* (termo que o ACS utiliza para se referir ao conjunto de usuário e conta). A operação pode ser uma das seguintes:

- `NONE`: Rejeita o *login* se o usuário não for identificado e ignora diferenças nos atributos;
- `CREATE`: Cria usuário e/ou conta se não forem identificados;
- `UPDATE`: Rejeita o *login* se o usuário não for identificado. Em caso de contradições, atualiza usuário e/ou conta para refletir os atributos recebidos do IdP;
- `CREATEANDUPDATE`: Cria usuário e/ou conta se não forem identificados. Em caso de contradições, atualiza usuário e/ou conta para refletir os atributos recebidos do IdP.

Em extensões futuras, mais tipos de operação podem ser adicionados para permitir a flexibilização da criação de domínios e *roles*, por exemplo. Todavia, devido à natureza de *login* desta comunicação com o IdP, a operação de *UserAccounts* servirá como escopo para os outros tipos de operação (por exemplo, não deverá ser possível criar um domínio sem criar uma conta e usuário).

4.2. Mapeamento

Desde a versão 4.18, lançada em 2023, o ACS permite definir a contabilização de recursos por *scripts* JavaScript com variáveis de ambiente pré-estabelecidas, chamados de regras de ativação[1]. O mesmo interpretador é também utilizado em seletores de armazenamento desde a versão 4.19[2].

Este trabalho reaproveita a funcionalidade para que o operador realize o mapeamento dos atributos recebidos do IdP (injetados no *script* através do objeto "`idp`") para uma estrutura pré-definida, representada por um objeto JavaScript que deve ser retornado ao fim da execução.

Este objeto retornado deve conter determinados atributos, especificados na lista a seguir. Dependendo das operações definidas, diferentes combinações de atributos podem ser obrigatórias. No caso de conflitos, o atributo "`uuid`" terá precedência sobre outros identificadores de um mesmo objeto.

- `user`:
  - `email`: Obrigatório para criação de usuário/conta.
  - `firstname`: Obrigatório para criação de usuário/conta.
  - `lastname`: Obrigatório para criação de usuário/conta.
  - `username`: Obrigatório para criação de usuário/conta e para *login*.
  - `uuid`: Opcional. Se presente, todos os outros atributos, incluídos os de `account` e `domain`, deixam de ser obrigatórios para *login*.
  - `timezone`: Opcional.
- `account`:
  - `accountname`: Opcional.

[1] https://github.com/apache/cloudstack/pull/5909 (Quota custom tariffs)
[2] https://github.com/apache/cloudstack/pull/7659 (Add dynamic secondary storage selection)

  - `details`: Opcional. Os atributos são no formato de nome da configuração e o respectivo valor.
    * `config.exemplo`: valor
    * ...
  - `networkdomain`: Opcional.
  - `role`:
    * `name`: Obrigatório para criação de conta.
    * `type`: Obrigatório para criação de conta se houver mais de uma *role* com o mesmo `name`.
    * `uuid`: Opcional. Se presente, `name` e `type` deixam de ser obrigatórios para criação de conta.
  - `uuid`: Opcional. Se presente e a conta existir, `domain` deixa de ser obrigatório para *login* e criação de usuário.
- `domain`:
  - `path`: Obrigatório para criação de usuário/conta e para *login*.
  - `uuid`: Opcional. Se presente, `path` deixa de ser obrigatório.
- `legacy`: Opcional. Se presente, indica que a política deve tentar realizar o *login* de um mapeamento que retorne apenas "`user.username`".

Excepcionalmente, para a manutenção da compatibilidade, o mapeamento pode conter apenas o atributo `user.username`. Se o *script* retornar apenas o valor de `user.username`, além de `legacy` = *true*, o ACS desempenhará o fluxo antigo.

4.3. Alterações

Para armazenamento das políticas de sincronização, será criada uma nova tabela `idp_sync_policy` no banco de dados do ACS, com as colunas descritas na Tabela 1. Serão criados os *endpoints* da API `createIdpSyncPolicy`, `updateIdpSyncPolicy`, `removeIdpSyncPolicy` e `listIdpSyncPolicies` para criar, atualizar, remover e listar as políticas de sincronização, respectivamente.

Para facilitar o processo de criação dos *scripts* de mapeamento, também será implementado o *endpoint* `validateIdpSyncPolicyMapping`, que realizará a simulação do processo de *login*, sem realizar alterações no banco de dados.

As configurações globais `idp.sync.policy.mapping.timeout` e `idp.sync.policy.auto.create.default.policy` serão adicionadas. A primeira determinará o tempo máximo de espera pela conclusão do processamento do mapeamento de uma política de sincronização. A segunda, habilitará a criação automática de uma política padrão quando o ACS encontrar um novo IdP.

5. Implementação

De acordo com a forma que o ACS é estruturado, foram criadas as representações `IdpSyncPolicy` e `IdpSyncPolicyVO`, bem como o objeto de acesso (*Data Access Object* – DAO) `IdpSyncPolicyDAO` para abstração do relacionamento com o banco de dados referente às políticas de sincronização.

Além destes, foi estabelecido o componente `IdpSyncPolicyManager`, que é chamado pelo *plugin* SAML depois de processar a resposta SAML e coletar os atributos enviados pelo IdP, para executar a política apropriada.

Coluna	Tipo	Descrição
id	bigint(20) unsigned	Chave primária, ID interno.
uuid	varchar(40)	Chave única, UUID externo.
idp_id	text	URL do IdP.
mapping	text	*Script* de mapeamento dos atributos recebidos do IdP aos atributos da entidade do ACS.
useraccount_operation	enum('NONE', 'CREATE', 'UPDATE', 'CREATEANDUPDATE')	Operação a ser realizada sobre usuário e conta. Valor padrão é "NONE".
description	text	Descrição opcional.
created	datetime	*Timestamp* de criação.
updated	datetime	*Timestamp* da última atualização.
update_count	bigint(20) unsigned	Número de atualizações.
removed	datetime	*Timestamp* de remoção.
removal_reason	text	Motivo para remoção.

Table 1. Colunas da tabela `idp_sync_policy`

Foram criadas algumas classes intermediárias (`AttributeMap`, `UserMap`, `AccountMap`, `DomainMap`, `RoleMap`) para modelar os atributos a serem recebidos. A principal desvantagem desta abordagem de acoplamento é que, quando alteradas as classes de usuário, domínio, conta, *role*, etc., pode ser necessário também alterar estas classes de mapeamento.

Em contrapartida, o benefício é encontrado no isolamento do funcionamento interno do banco do ACS, reutilizando as mesmas checagens que ocorrem no fluxo mais comum de operações através dos *endpoints* diretos, como *createAccount*, *updateUser*, etc. Assim, não apenas os valores dos atributos são validados, como também o estado do usuário dentro do ACS (se está desabilitado ou bloqueado, por exemplo) também é respeitado.

Para realizar a adaptação para o novo modelo, foi adicionada uma verificação à inicialização do *plugin* SAML. Ela ocorre durante a análise dos metadados dos IdPs, se a configuração `idp.sync.policy.auto.create.policy` for verdadeira. Para cada IdP adicionado, o ACS procura por uma política de sincronização ativa daquele IdP. Se não encontrada, uma padrão é criada.

Foram criadas as classes referentes aos comandos de API (`CreateIdpSyncPolicyCmd`, `UpdateIdpSyncPolicyCmd`, `RemoveIdpSyncPolicyCmd`, `ListIdpSyncPoliciesCmd` e `ValidateIdpSyncPolicyMappingCmd`), que definem os parâmetros a serem recebidos na chamada de API e chamam o gerenciador. Os comandos só serão acessíveis e adicionados à lista de comandos se o *plugin* SAML2 estiver habilitado (`saml2.enabled`).

O comando de *login SAML*, `SAML2LoginAPIAuthenticatorCmd`, foi modificado para coletar todos os atributos recebidos ao invés de procurar apenas o nome de usuário descrito na configuração `saml2.user.attribute`. Uma vez coletados, eles são encaminhados ao gerenciador das políticas de sincronização, `IdpSyncPolicyManager`.

6. Conclusão

A partir dos conceitos da literatura relacionados aos temas de gestão de identidade e computação em nuvem e do processo de diagramação do fluxo atual de comunicação SAML do ACS, propusemos uma abordagem denominada Política de Sincronização para determinar como o ACS deve lidar com os atributos recebidos em uma resposta SAML.

A fim de iniciar uma discussão na comunidade do ACS em torno dos temas de autenticação e autorização, a abordagem foi implementada para possibilitar que o ACS delegue a gerência das identidades do sistema, criando e atualizando as suas identidades quando deparado com informações novas vindas do provedor de identidade.

Apesar dos limites presentes no escopo atual da implementação, o trabalho espera abrir caminho para futuras inovações e melhorias no processo de integração entre o Apache CloudStack e os provedores de identidade, sugerindo, a seguir, novas possibilidades para a evolução da plataforma.

6.1. Trabalhos futuros

Algumas das melhorias previstas em volta da funcionalidade apresentada neste trabalho incluem:

- Integração com outros protocolos de SSO, como OAuth e OIDC;
- Gerenciamento das políticas de sincronização pela UI;
- Restrição de criação e atualização de contas por IP;
- Criação e/ou atualização automática de *roles*, domínios e projetos;
- Restrição de permissões de acordo com os atributos recebidos, tal qual RBAC-A com foco em *roles* [Kuhn et al. 2010];
- Agregação de atributos de múltiplos IdPs [Chadwick and Inman 2009];
- Flexibilização ou restrição das permissões de acesso aos recursos em si, indo além de apenas a existência ou não de permissão de acesso ao *endpoint* da API;
- Restringir e/ou remover recursos de acordo com as informações do IdP.

Uma funcionalidade de alta complexidade, mas que poderia trazer diversos benefícios é possibilitar que o ACS atue como um IdP. Além de permitir a participação em federações de identidade, também resultaria em melhor integração de ambientes distribuídos em diferentes regiões, permitindo a centralização da gestão de identidades sem a necessidade de *software* terceiro.

Outra importante possibilidade de trabalho futuro seria um modo de proporcionar controle ao usuário sobre seus atributos quando recebidos pelo SP (neste caso, o ACS) [Weingärtner 2014].

References

ASF (2024a). *Apache CloudStack*. The Apache Software Foundation.

ASF (2024b). *Managing Accounts, Users and Domains*. The Apache Software Foundation.

Bertino, E. and Takahashi, K. (2011). *Identity Management*. Artech House, Norwood, MA.

Chadwick, D. W. (2009). *Federated Identity Management*, pages 96–120. Springer Berlin Heidelberg, Berlin.

Chadwick, D. W. and Inman, G. (2009). Attribute aggregation in federated identity management. *Computer*, 42(5):33–40.

ITU-T (2009). Ngn identity management framework. *Recommendation ITU-T Y.2720*.

Kuhn, D. R., Coyne, E. J., and Weil, T. R. (2010). Adding attributes to role-based access control. *Computer*, 43(6):79–81.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. *Special Publication 800-145*.

Weingärtner, R. (2014). Controle de disseminação de dados sensíveis em ambientes federados.

Wilson, Y. and Hingnikar, A. (2019). *Solving Identity Management in Modern Applications*. Apress, New York.