OpenStack Identity Federation

Many thanks to:

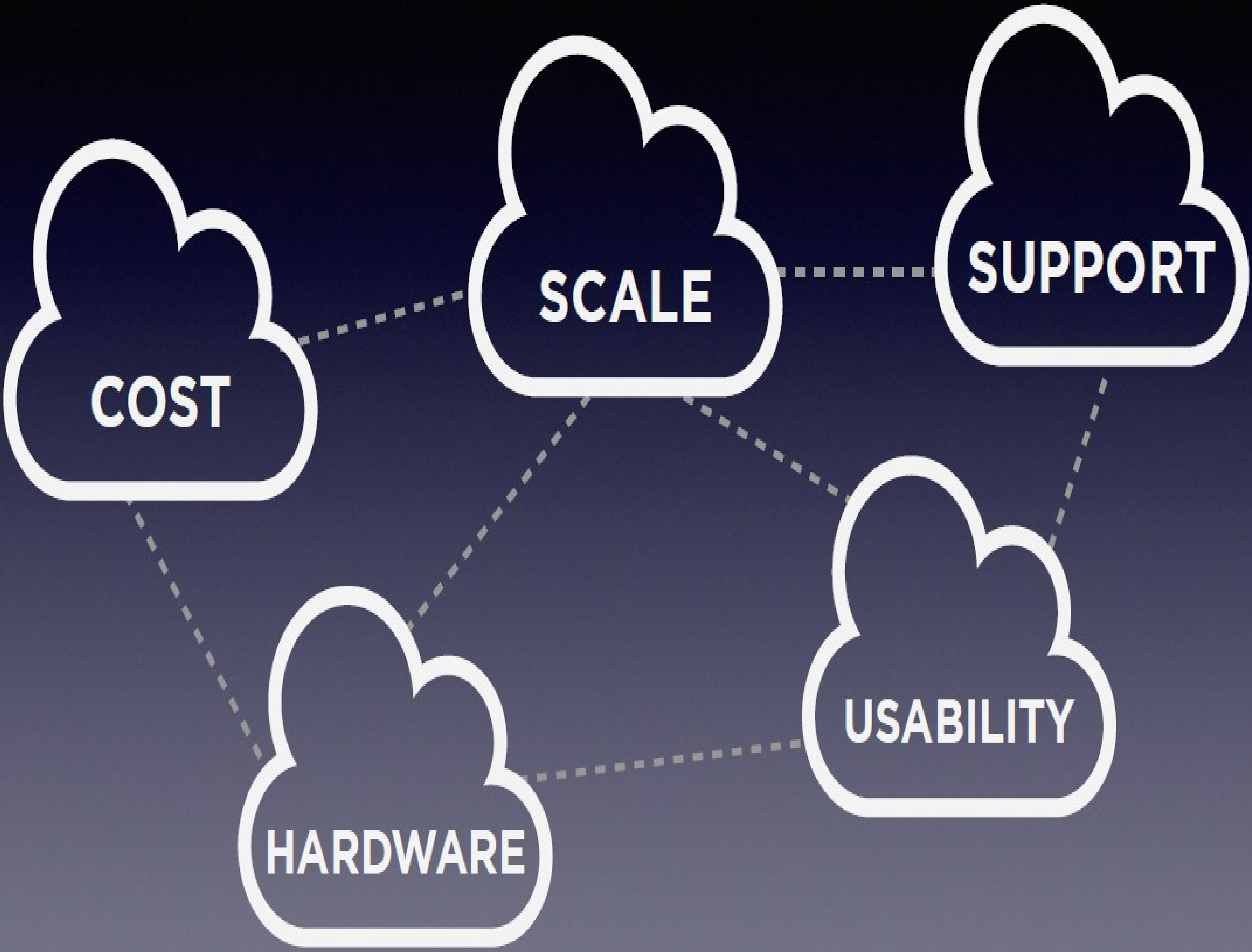
Who are we?

Joe Savak – Rackspace Product Manager (° ͜ʖ°)

Brad Topol – IBM Distinguished Engineer ¯_(ツ)_/¯

Jorge Williams – Rackspace Principal Architect ●~*

Steve Martinelli – IBM Software Developer, Keystone Core (╯°□°)╯ ┻━┻


SCALE
SUPPORT
COST
USABILITY
HARDWARE

Feder**what**ion?

The mechanisms to establish trusts between identity providers and OpenStack clouds enabling a user to:

- Securely access resources (servers, files, volumes, dbs. unicorns, etc)
- Across multiple endpoints provided in multiple authorized clouds
- With their single credential
- Maintained in a trusted identity provider
- Without having to provision additional identities or re-login all the time

(ADFS, Tivoli Federated IdM, Shibboleth are other federated identity implementations)

Feder**why**tion?

1. Any time a new identity is provisioned, it is a security risk.

2. It is a burden on clients to deal with multiple tokens across multiple cloud service providers.

3. We spend too much time logging in or going through forget password workflows.

4. We spend too much time administrating identities in various service providers. (Imagine administration of a full university or enterprise with identities always in flux)

5. The **best test of interoperability in the cloud** is to enable one identity access across multiple clouds.

6. Removes a blocker to cloud brokering and multi-cloud workload management.

SAML, OpenID Connect, AbFab?

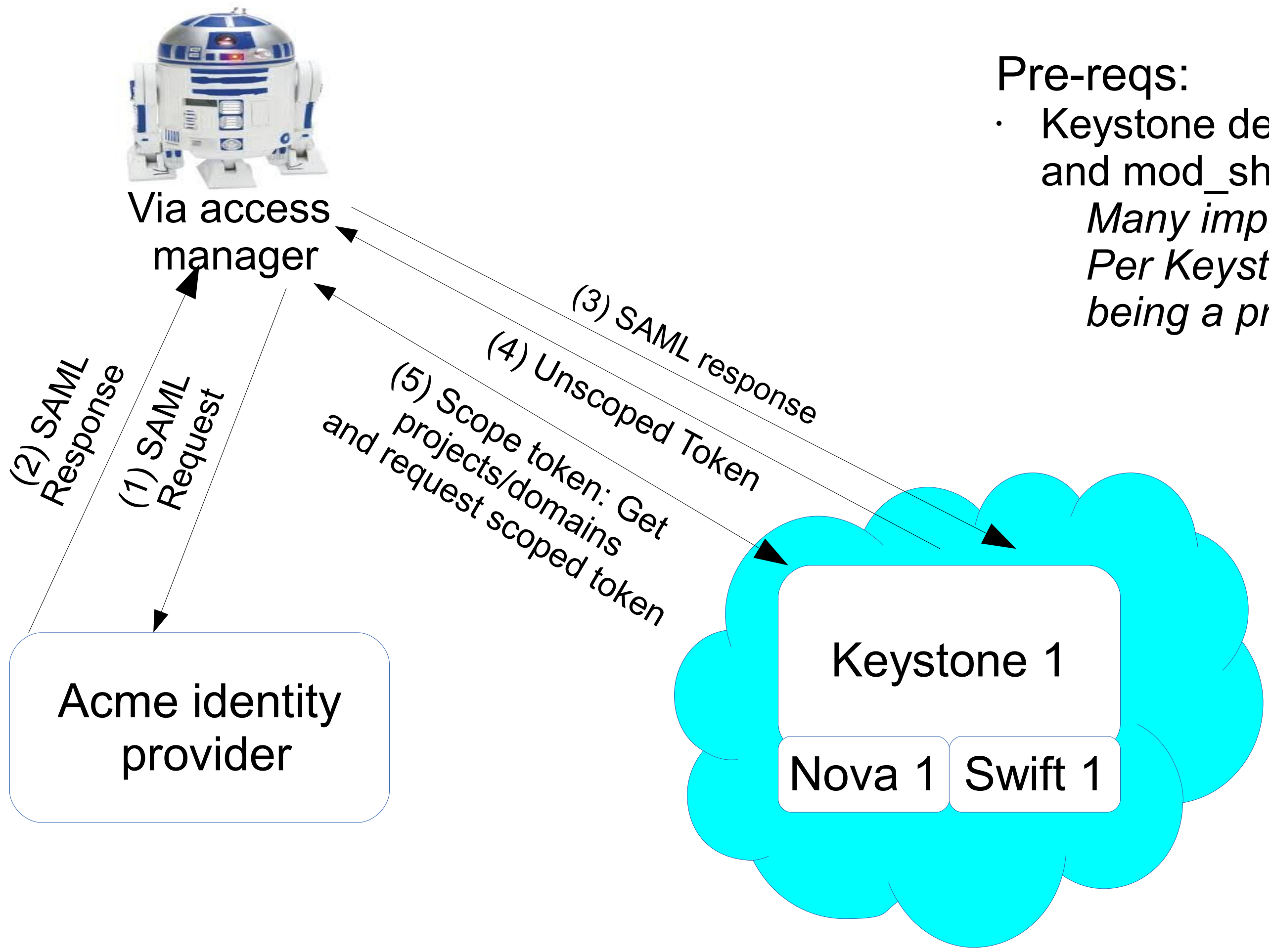
There are many federation protocols out there.

Each with their own pros & cons.

So keystone federation was made to be **extensible to allow various protocol support**.

- SAML2 ← supported in icehouse, determined at HK summit to support this first. As part of icehouse, we use Shibboleth to avoid re-inventing the wheel for the SAML authentication handshake.
- OpenID connect (based on OAuth2)← in the works, Juno
- AbFab ← accepting pull requests.

The big picture – Icehouse – federation with identity providers



Pre-reqs:

- Keystone deployed as part of apache and mod_shib enabled for SAML 2 handshake
  Many implementors use Apache already.
  Per Keystone core: Keystone is focusing on being a proxy instead of a provider

With your company credentials, you can access resources and execute APIs in an OpenStack cloud without having to provision a new identity for that cloud.
Just keep using your company-issued identity!

R2D2 could be:

- an access manager like OneLogin or NetIQ Access Manager
- A federation-enabled openstack client (using Enhanced Client or Proxy, possibly)

both would need to speak the supported federation protocol for the cloud.

Feedback wanted!

Detailed use cases are always welcome :)

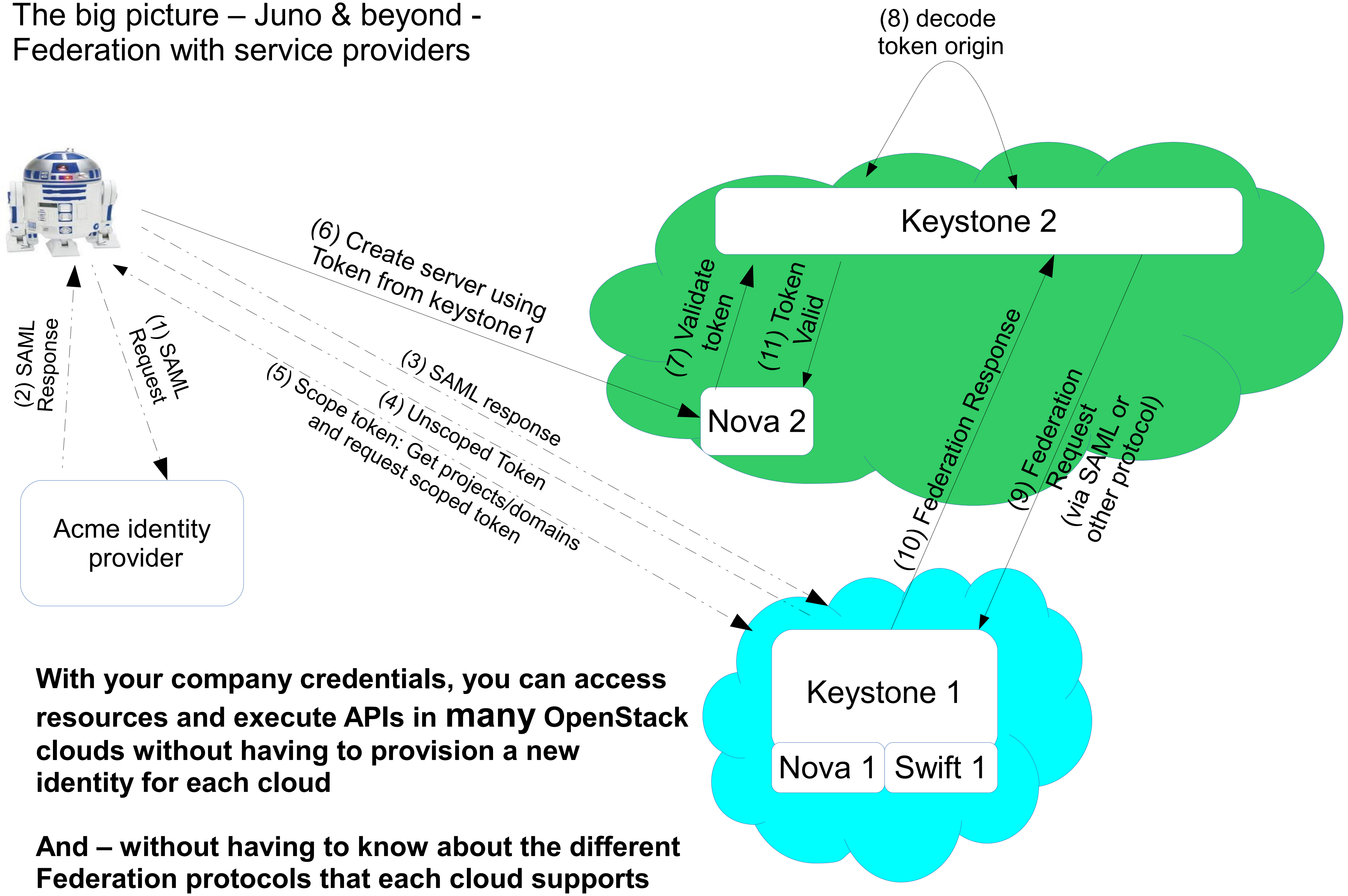
Contribute back to the community!

- Review the API spec
- Propose an API spec
- Review the code
- Review the docs
- Create a guide
- Play with the code
- Test the code with your development environment
- Etc...

Federated Identity is easily one of the **most visible blueprints the Keystone team has done in a while**, as a community, let's make sure it's awesome!

https://blueprints.launchpad.net/keystone/+spec/keystone-to-keystone-federation

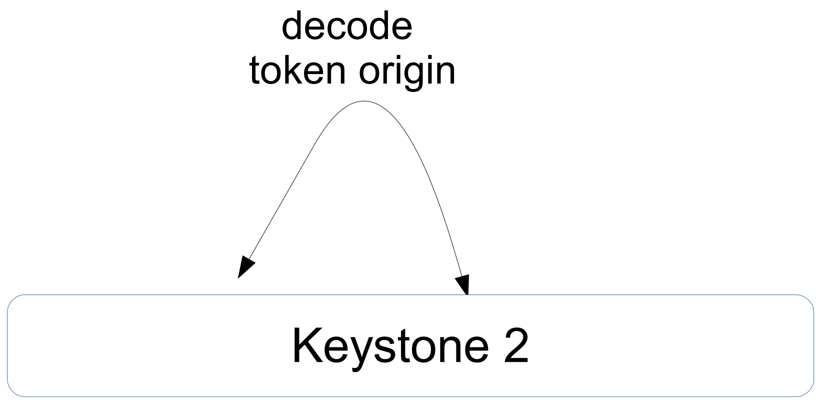
The big picture – Juno & beyond - Federation with service providers



With your company credentials, you can access resources and execute APIs in many OpenStack clouds without having to provision a new identity for each cloud

And – without having to know about the different Federation protocols that each cloud supports

Juno & beyond
Wait... decode token??



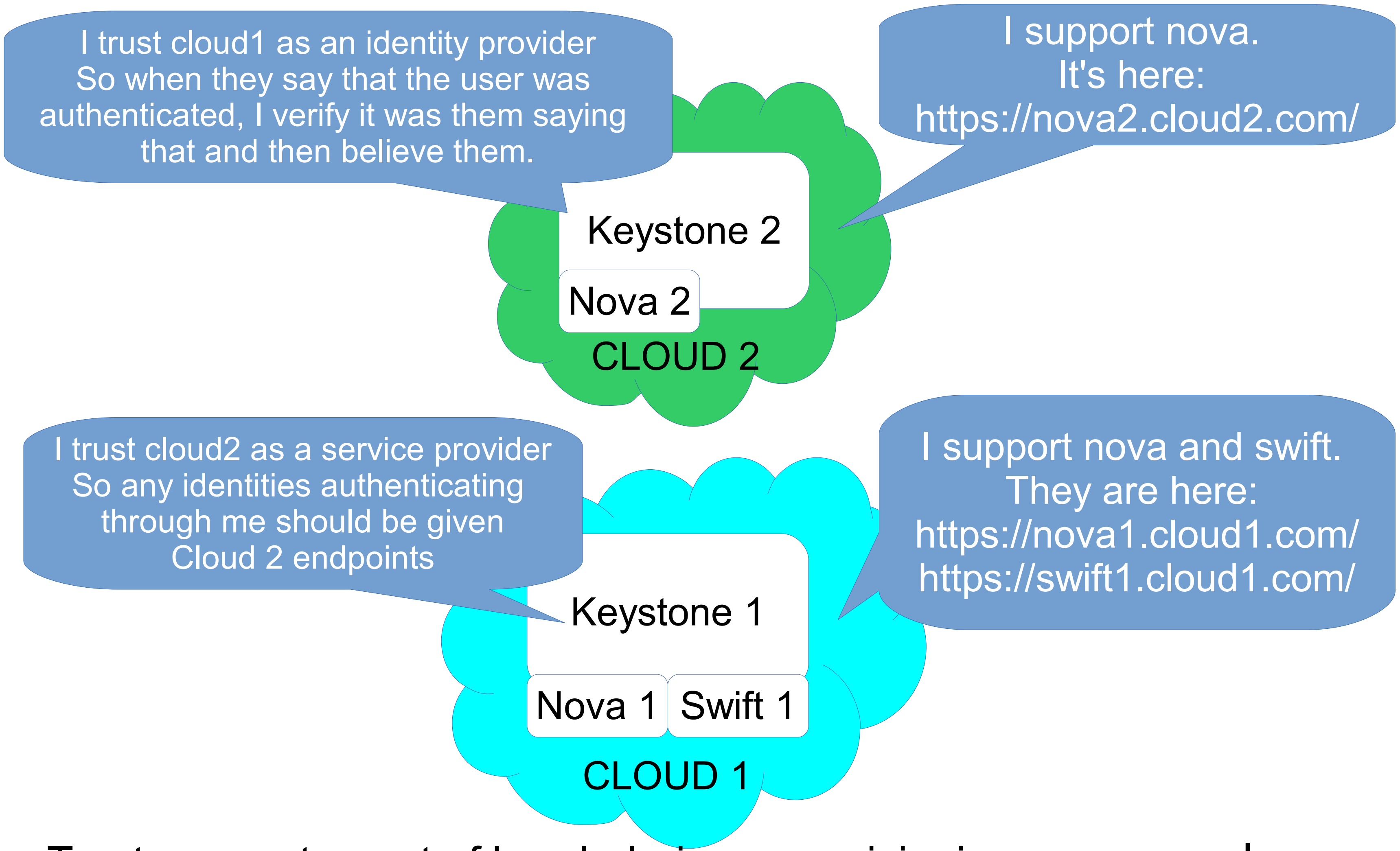
A federated token needs to include information about where the originating authentication occurred. This is needed for keystone to validate the asserted identity and understand what access the identity should have.

1 potential solution building off of what is supported in icehouse:
Token metadata includes: {originating-identity-provider aka issuer}/{protocol}/{subject}

Need to see how this works with PKI.

Allows auditing support across multiple clouds!
Potentially visible by emitting CADF events

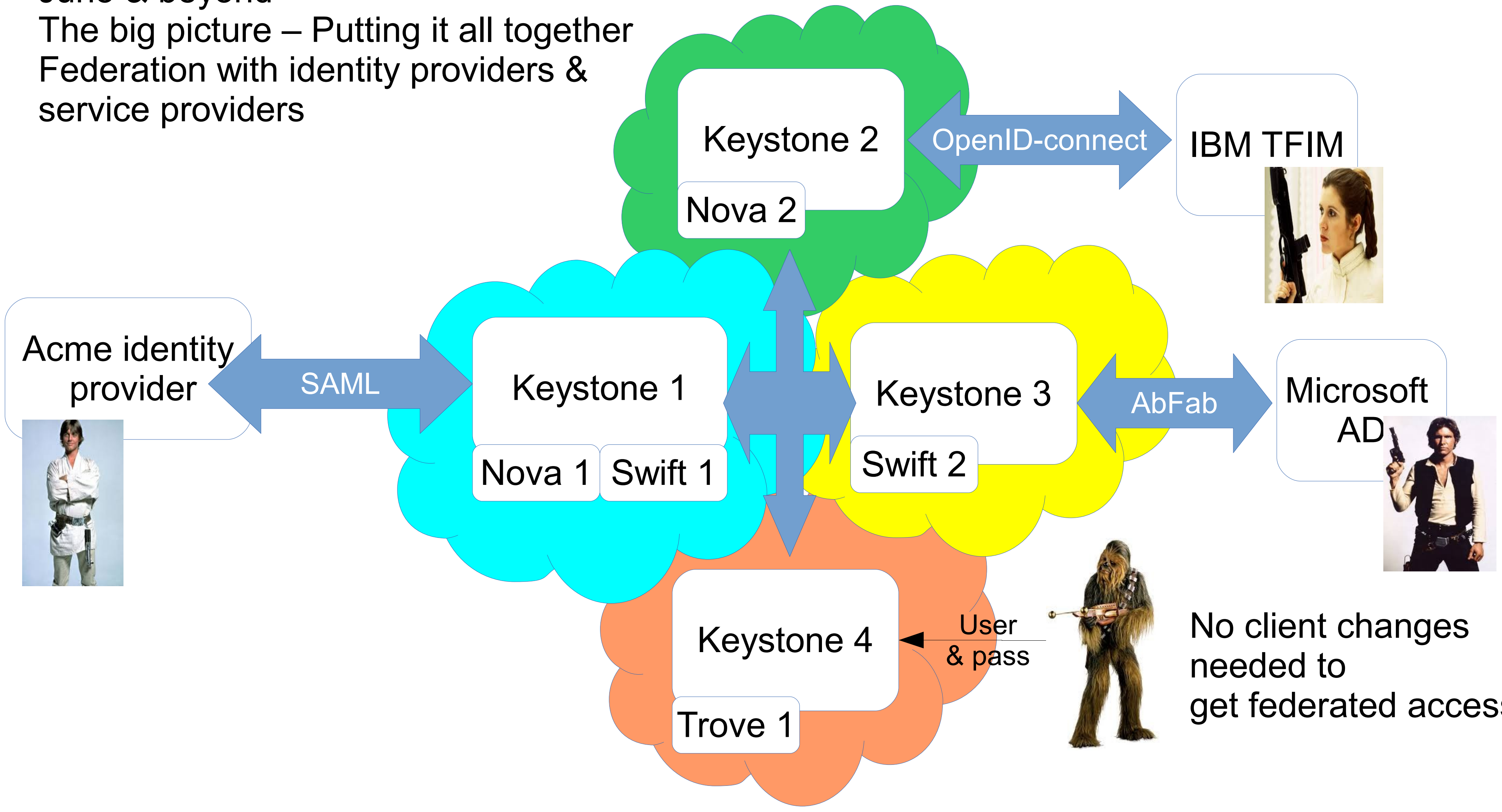
Juno & beyond
Wait – how do clouds trust each other?


I trust cloud1 as an identity provider
So when they say that the user was authenticated, I verify it was them saying that and then believe them.
I support nova.
It's here:
https://nova2.cloud2.com/
Keystone 2
Nova 2
CLOUD 2
I trust cloud2 as a service provider
So any identities authenticating through me should be given Cloud 2 endpoints
I support nova and swift.
They are here:
https://nova1.cloud1.com/
https://swift1.cloud1.com/
Keystone 1
Nova 1
Swift 1
CLOUD 1

Trusts are setup out of band, during a provisioning process where public keys are exchanged between two (or more) parties.

One service catalog is returned containing all accessible endpoints across clouds when a user authenticates

Juno & beyond
The big picture – Putting it all together
Federation with identity providers & service providers



No client changes needed to get federated access

What was delivered in Icehouse?

Officially, from the Icehouse release notes:

"The OS-FEDERATION extension allows Keystone to consume federated authentication via an Apache module for multiple Identity Providers, and mapping federated attributes into OpenStack group-based role assignments" – Dolph Mathews (Keystone PTL)

Outline:

- New Keystone OS-FEDERATION APIs
  - Identity Providers
  - Protocols
  - Mappings
- Motivation for Mappings
  - Setting up OpenStack Groups
  - SAML Assertions
  - Creating a Mapping
- Getting In!
  - Requesting an unscoped token
  - Listing available resources
  - Requesting a scoped token

~=

OPENSTACK SUPERUSER
MAREK DENIS / CERN
The CERN infrastructure team migrates an average of 100 servers per week to OpenStack compute nodes, ensuring their researchers have access to technology that enables their cutting-edge high energy partical physics research.

New Keystone OS-FEDERATION APIs

Identity Providers: `/OS-FEDERATION/identity_providers/{idp_id}`

- An Identity Provider is a third party service that is trusted by the Identity API to authenticate identities.
- Register SAML Identity Providers such as ADFS or Tivoli Federated Identity Manager.

```
{
    "identity_provider": {
        "description": "Stores ACME identities",
        "enabled": true,
        "id": "ACME",
    }
}
```

- description (string) - Describes the identity provider.
- enabled (boolean) - Indicates whether this identity provider should accept federated authentication requests.
- id (string) - User-defined unique id to identify the identity provider.

New Keystone OS-FEDERATION APIs

Protocols:
`/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol_id}`

- A Protocol entry contains information that dictates which mapping rules to use for a given incoming request. An IdP may have multiple supported protocols.
- Currently, only the SAML 2.0 federation protocol is supported. However, the framework is extensible to support other federation protocols, i.e.: OpenID, WS-Federation, SAML 1.0.
- Identity Providers can communicate in many protocols, so associate an ***Identity Provider*** with a ***Mapping***, based on a protocol.

```
{
    "protocol": {
        "id": "saml2",
        "mapping_id": "xyz234",
    }
}
```

- mapping_id (string) - Indicates which mapping should be used to process federated authentication requests.
- id (string) - User-defined unique id to identify the protocol.

New Keystone OS-FEDERATION APIs

Mappings: `/OS-FEDERATION/mappings/{mapping_id}`

- A mapping is a set of rules to map federation protocol attributes to Identity API objects. An Identity Provider can have a single mapping specified per Protocol. A Mapping is simply a list of rules.
- A mapping is a method to translate remote attributes (from an Identity Provider) to local attributes (Keystone entities).
- Mappings are created as a top level resource so as to enable re-use between Identity Providers.

```
{
    "mapping": {
        "id": "ACME_MAP",
        "rules": [...],
    }
}
```

More on this soon!

- rules (list) - Each object contains a rule for mapping attributes to Identity API concepts. A rule contains a remote attribute description and the destination local attribute.
- id (string) - User-defined unique id to identify the mapping.

Motivation for Mapping

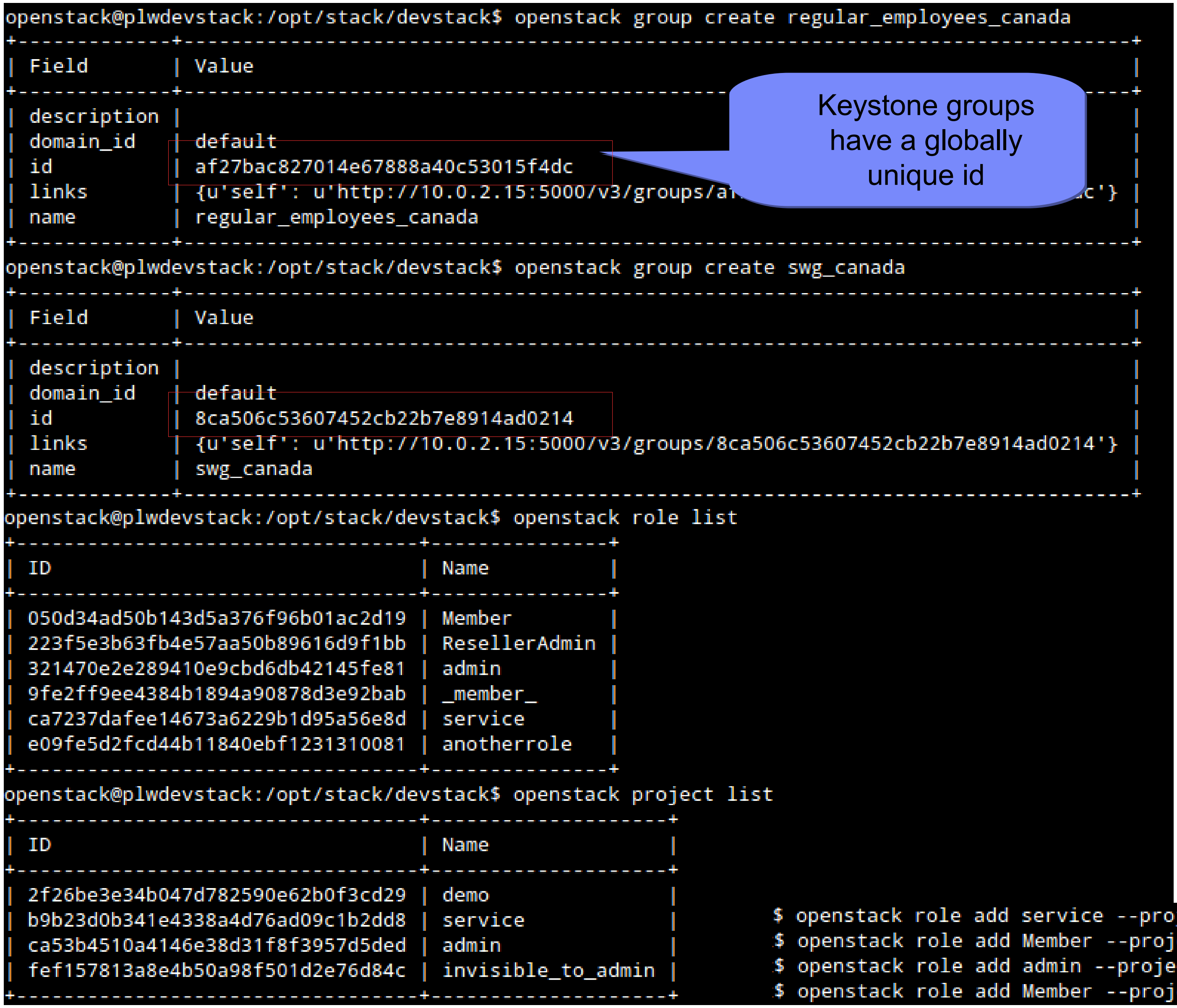
- Setting the scene:
  - Classic Keystone, does user **stevemar**, have role **developer**, on project **services**?
    - Or does the requesting user belong to **group** that has role **developer**, on project **services**?
  - Admin means something different to CNN than it does to Coca Cola Co.
    - Neither mean anything in Keystone for federated users.

- Identifying the problem:
  - Federated users do not exist in Keystone, they exist on an Identity Provider.
  - An Identity Provider will only return attributes related to an identity.
  - We don't have a solution for mapping what an IdP sees to what Keystone knows.

- Finding a solution:
  - Create a mapping to handle the translation and establish relationships between Keystone attributes and Identity Provider attributes.
  - Initially, a mapping would be a 1:1 relationship between Identity Provider attributes and Keystone groups.
  - A federated user can authenticate with an Identity Provider, and be mapped to a Keystone group, and will **inherit** the roles from the group.

Setting up OpenStack for Groups

- Create groups, that have a role(s) on a project or domain.
- Can be done via CLI for convenience.



```
openstack@plwdevstack:/opt/stack/devstack$ openstack group create regular_employees_canada
+-------------+------------------------------------------------------------------------------+
| Field       | Value                                                                        |
+-------------+------------------------------------------------------------------------------+
| description |                                                                              |
| domain_id   | default                                                                      |
| id          | af27bac827014e67888a40c53015f4dc                                             |
| links       | {u'self': u'http://10.0.2.15:5000/v3/groups/af                           c'} |
| name        | regular_employees_canada                                                     |
+-------------+------------------------------------------------------------------------------+
openstack@plwdevstack:/opt/stack/devstack$ openstack group create swg_canada
+-------------+------------------------------------------------------------------------------+
| Field       | Value                                                                        |
+-------------+------------------------------------------------------------------------------+
| description |                                                                              |
| domain_id   | default                                                                      |
| id          | 8ca506c53607452cb22b7e8914ad0214                                             |
| links       | {u'self': u'http://10.0.2.15:5000/v3/groups/8ca506c53607452cb22b7e8914ad0214'} |
| name        | swg_canada                                                                   |
+-------------+------------------------------------------------------------------------------+
openstack@plwdevstack:/opt/stack/devstack$ openstack role list
+----------------------------------+---------------+
| ID                               | Name          |
+----------------------------------+---------------+
| 050d34ad50b143d5a376f96b01ac2d19 | Member        |
| 223f5e3b63fb4e57aa50b89616d9f1bb | ResellerAdmin |
| 321470e2e289410e9cbd6db42145fe81 | admin         |
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_      |
| ca7237dafee14673a6229b1d95a56e8d | service       |
| e09fe5d2fcd44b11840ebf1231310081 | anotherrole   |
+----------------------------------+---------------+
openstack@plwdevstack:/opt/stack/devstack$ openstack project list
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 2f26be3e34b047d782590e62b0f3cd29 | demo               |
| b9b23d0b341e4338a4d76ad09c1b2dd8 | service            |
| ca53b4510a4146e38d31f8f3957d5ded | admin              |
| fef157813a8e4b50a98f501d2e76d84c | invisible_to_admin |
+----------------------------------+--------------------+
```

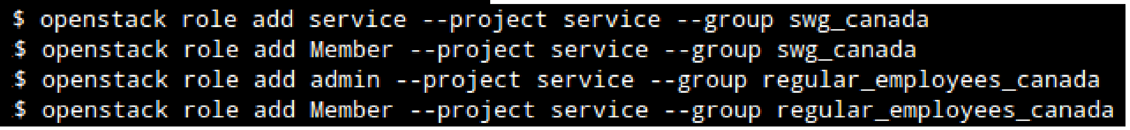
Keystone groups have a globally unique id

Keystone Group IDs:

- regular_employees_canada
  - (af27ba … 15f4dc)
- swg_canada
  - (8ca506 … ad0214)

Associate **groups** with a **role** on a **project.**

- (Roles and projects listed on left side)



```
$ openstack role add service --project service --group swg_canada
$ openstack role add Member --project service --group swg_canada
$ openstack role add admin --project service --group regular_employees_canada
$ openstack role add Member --project service --group regular_employees_canada
```

SAML Assertions

- An assertion from a valid Identity Provider, is an indication that the user has been authenticated.
- A snippet from a SAML assertion is seen below, containing user and group information (from an IdP perspective).
- The '**idp_group**' in the SAML attributes are the IdPs method of assigning groups, these need to be mapped back to the groups that were created in the previous step.

```
<saml:AttributeStatement>
    <saml:Attribute Name="subject">
        <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
                             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                             xsi:type="xs:string"
                             >stevemar</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="idp_group">
        <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
                             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                             xsi:type="xs:string"
                             >IBM Regular Employees Canada</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute Name="idp_group">
        <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
                             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                             xsi:type="xs:string"
                             >SWG Canada</saml:AttributeValue>
    </saml:Attribute>
</saml:AttributeStatement>
```

IdP Group attributes:

- IBM Regular Employees Canada
- SWG Canada

Keystone Group IDs:

- regular_employees_canada
  - (af27ba … 15f4dc)
- swg_canada
  - (8ca506 … ad0214)

Adding the Mapping

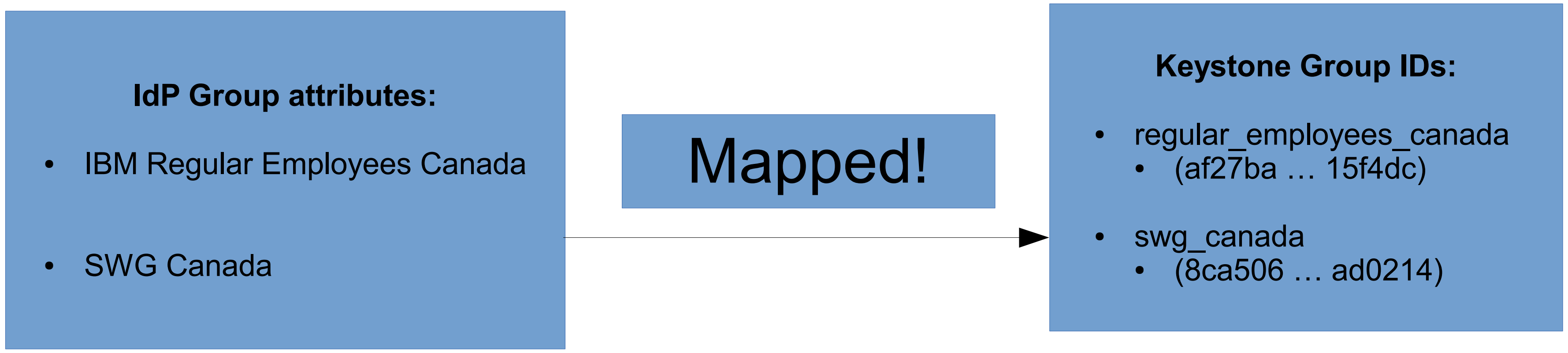
- Create a mapping to map the IdP attributes to Keystone attributes.
- Example request body sent to `OS-FEDERATION/mappings/BP_MAP`

```
{
  "mapping": {
    "rules": [
    {
      "local": [
      {
        "group": {
          "id": "af27bac827014e67888a40c53015f4dc"
          }
        }
      ],
      "remote": [
      {
        "type": "idp_group",
        "any_one_of": [
          "IBM Regular Employees Canada"
        ]
      }
      ]
     }
    ]
  }
}
```

In this case 'IBM Regular Employees Canada' maps to ID 'af27ba … 15f4dc'

```
"rules": [
{
  "local": [{
    "user": {
      "name": "{0}"
    }}],
    "remote": [{
      "type": "subject"
    }]
},
{
  "local": [{
    "group": {
      "id": "8ca506c53607452cb22b7e8914ad0214"
}}],
  "remote": [{
    "type": "idp_group",
    "any_one_of": [
      "SWG Canada"
    ]}]
  }
]
```

A mapping can have many rules!



Getting In!

- Setting the scene:
  - The user is already authenticated through their own Identity Provider.
  - Keystone is acting as a proxy, for them to have access to the cloud.
  - In classic Keystone authentication, scope is defined as a project or domain the user wishes to access.
    - That doesn't change with federation.

- Identifying the problem:
  - The federated user doesn't know anything about the resources (projects/domains) available.

- Finding a solution:
  - Create new APIs to allow a look-up
  - How to get in.
    - Initially retrieve an unscoped token.
    - Look up which resources (projects/domains) are available.
    - Retrieve a scoped token.
  - Success!

Getting In! - Part 1

Request an unscoped token:

- `/OS-FEDERATION/identity_providers/{idp_id}/protocols/{protocol}/auth`

  - Initiate the SAML handshake.
  - A protected URL, as such a request made to the URL would be redirected to the Identity Provider, to start the SAML authentication procedure.
    - Really controlled by Apache through mod_shib, for now.
  - The returned token would look like a normal unscoped Keystone token, but with extra Federation content.

  Response Header: `X-Auth-Token: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

  - **Response Body:**

```
{
    "token": {
        "methods": [
            "saml2"
        ],
        "user": {
            "id": "stevemar",
            "name": "stevemar",
            "OS-FEDERATION": {
                "identity_provider": "BP",
                "protocol": "SAML",
                "groups": [
                    {"id": "af27ba … 15f4dc"},
                    {"id": "8ca506 … ad0214"}
                ]
            }
        }
    }
}
```

Standard Keystone token ID (PKI, UUID …)

Data from mapping output

Data about the IdP and protocol.

Data from mapping output

Getting In! - Part 2

By using group memberships, a federated user can have access to a resource (project or domain).

Use the unscoped token returned from the previous step and call either:

List projects a federated user can access:

- `GET /OS-FEDERATION/projects`

`OR`

List domains a federated user can access:

- `GET /OS-FEDERATION/domains`

Output will be in the same format as `GET /v3/projects` or `GET /v3/domains`

Getting In! - Part 3

- **Request a scoped token:** `/auth/tokens`

- Once a federated user knows the project or domain id, a request can be made to retrieve a token that has access to that specific project or domain.
- The returned token should look like a regular Keystone token!

Request Body:

```
{
  "auth": {
    "identity": {
      "methods": [
        "saml2"
      ],
      "saml2": {
        "id":
"wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
      }
    }
  },
  "scope": {
    "project": {
      "id": "263fd9"
    }
  }
}
```

Unscoped Token ID taken from Step 1

Taken from Step 2

Response Body:

```
{
  "token": {
    "methods": ["saml2"],
    "roles": [
      { "id": "050d34ad50b143d5a376f96b01ac2d19",
        "name": "Member" },
      { "id": "ca7237dafee14673a6229b1d95a56e8d",
        "name": "service" }
    ],
    "expires_at": "2014-03-28T03:07:42.027427Z",
    "project": {
      "domain": {
        "id": "default",
        "name": "Default"
      },
      "id": "b9b23d0b341e4338a4d76ad09c1b2dd8",
      "name": "service"
    },
    "user": {
      "id": "joeuser%40ca.ibm.com",
      "name": "joeuser@ca.ibm.com"
    },
    "issued_at": "2014-03-28T02:07:42.027492Z"
  }
}
```

Our roles on the project, inherited from groups

SUCCESS!!!!

Federation Design Sessions

Additional design sessions related to Federation:

- Federation – TODAY! - 1:30 pm – 2:10 pm - B306
- Locally managed identities – TODAY! • 2:20 pm – 3:00 pm - B306
- User & Group IDs – TOMORROW! • 11:40 am – 12:20 pm - B306