ARC407

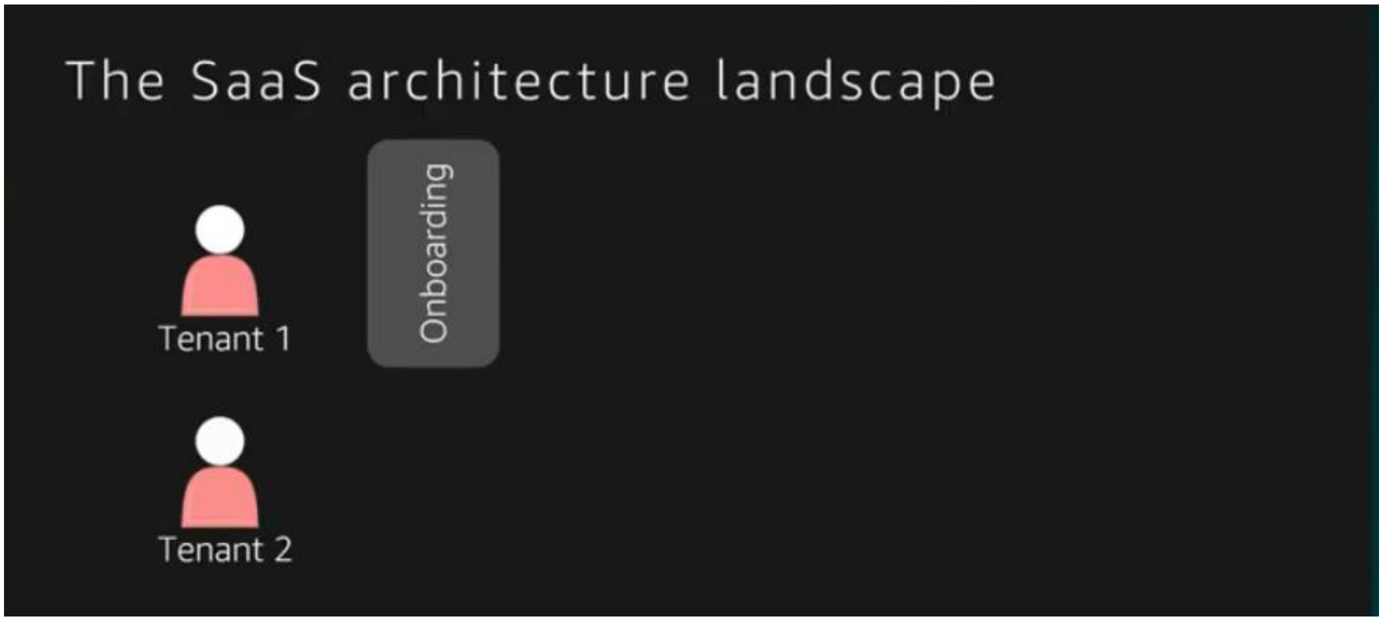
AWS re:INVENT

Deconstructing SaaS: A Deep Dive into Building Multi-tenant Solutions on AWS

Tod Golding, Partner Solutions Architect, SaaS Global Tech Lead
todg@amazon.com

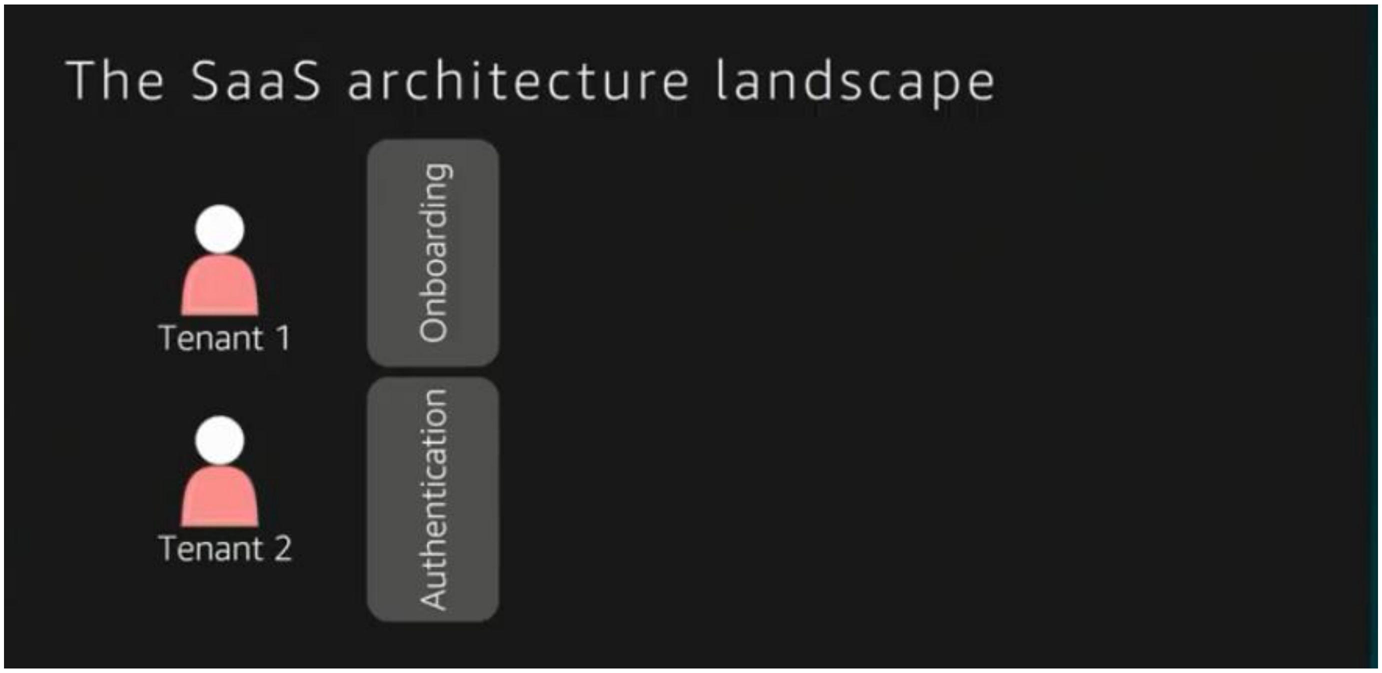
December 1, 2017

SaaS presents developers with a unique blend of architectural challenges. While the concepts of multi-tenancy are straightforward, the reality of making all the moving parts work together can be daunting. In this session, we move beyond the conceptual bits of SaaS and look under the hood of an SaaS application. Our goal is to examine the fundamentals of identity, data partitioning, and tenant isolation through the lens of a working solution and to highlight the challenges and strategies associated with building a next generation SaaS application on AWS. We look at the full lifecycle of registering new tenants, applying security policies to prevent cross-tenant access, and leveraging tenant profiles to effectively distribute and partition tenant data. We intend to connect many of the conceptual dots of an SaaS implementation, highlighting the tradeoffs and considerations that can shape your approach to SaaS architecture.

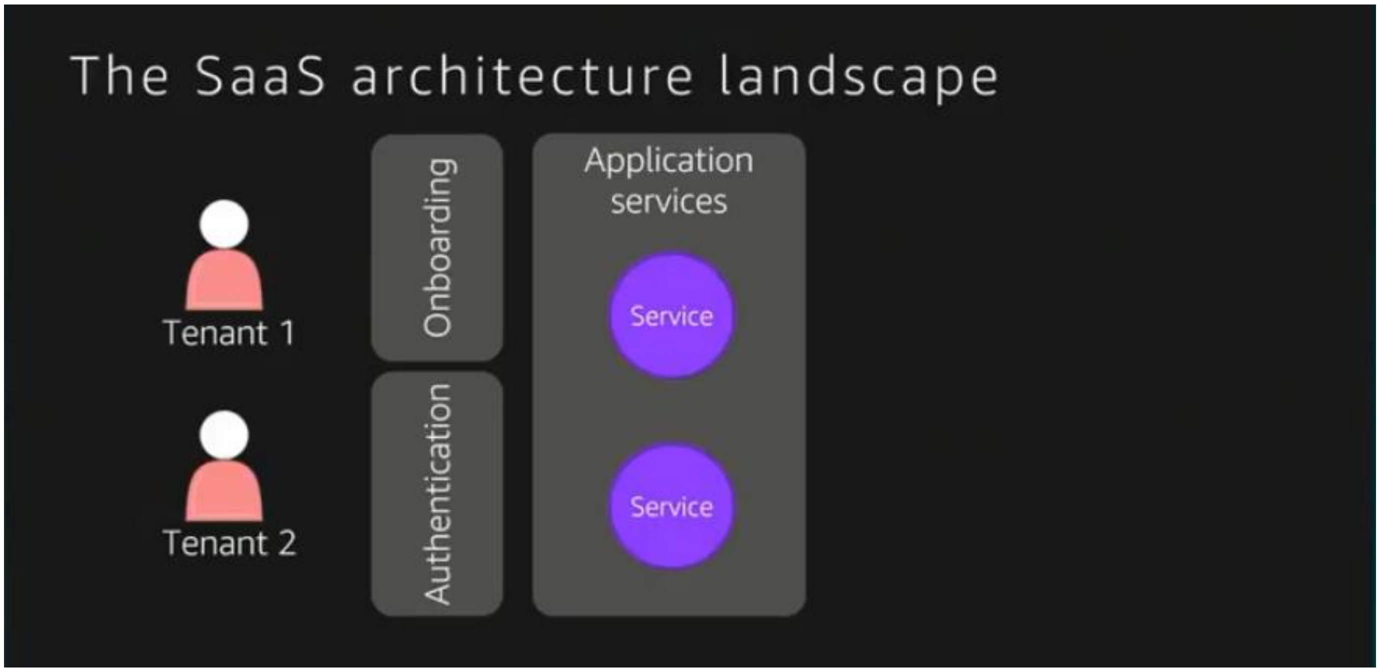


The SaaS architecture landscape

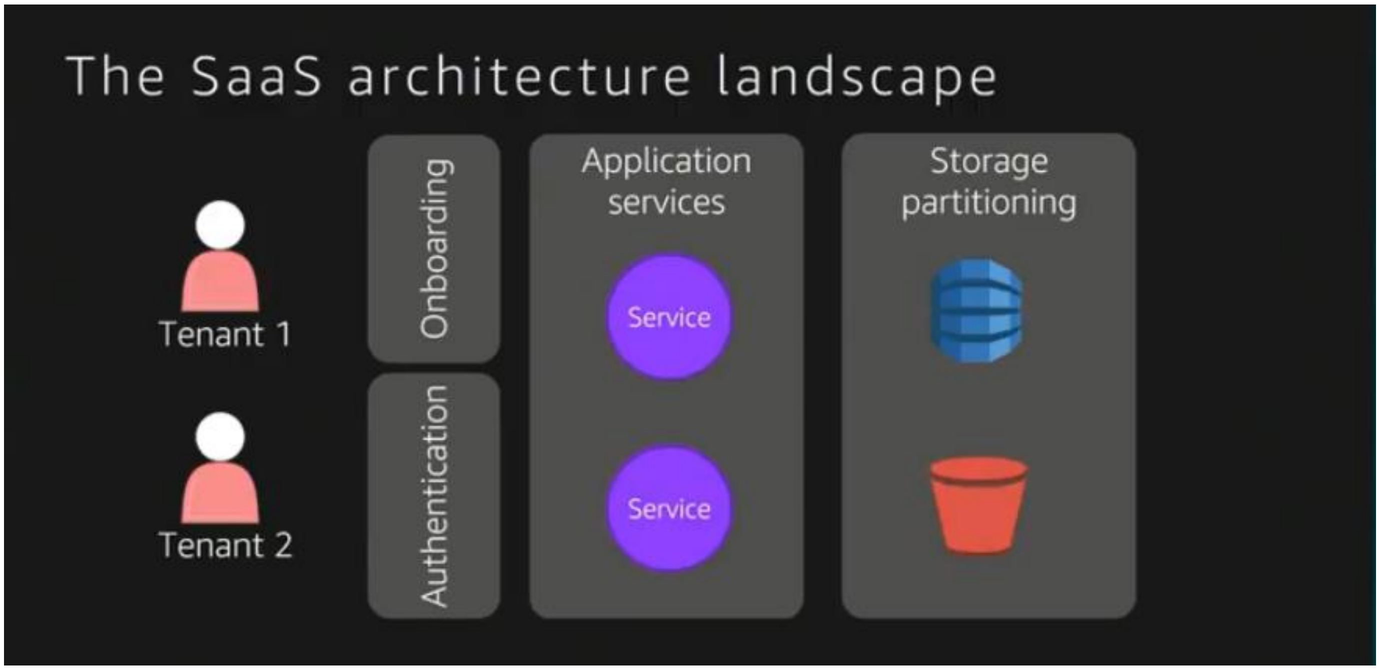
All the tenant data is shared in this architecture since this is a polled environment. Onboarding includes sign up and other things that have to be provisioned, spurn up, the policies that need to be created for the tenant



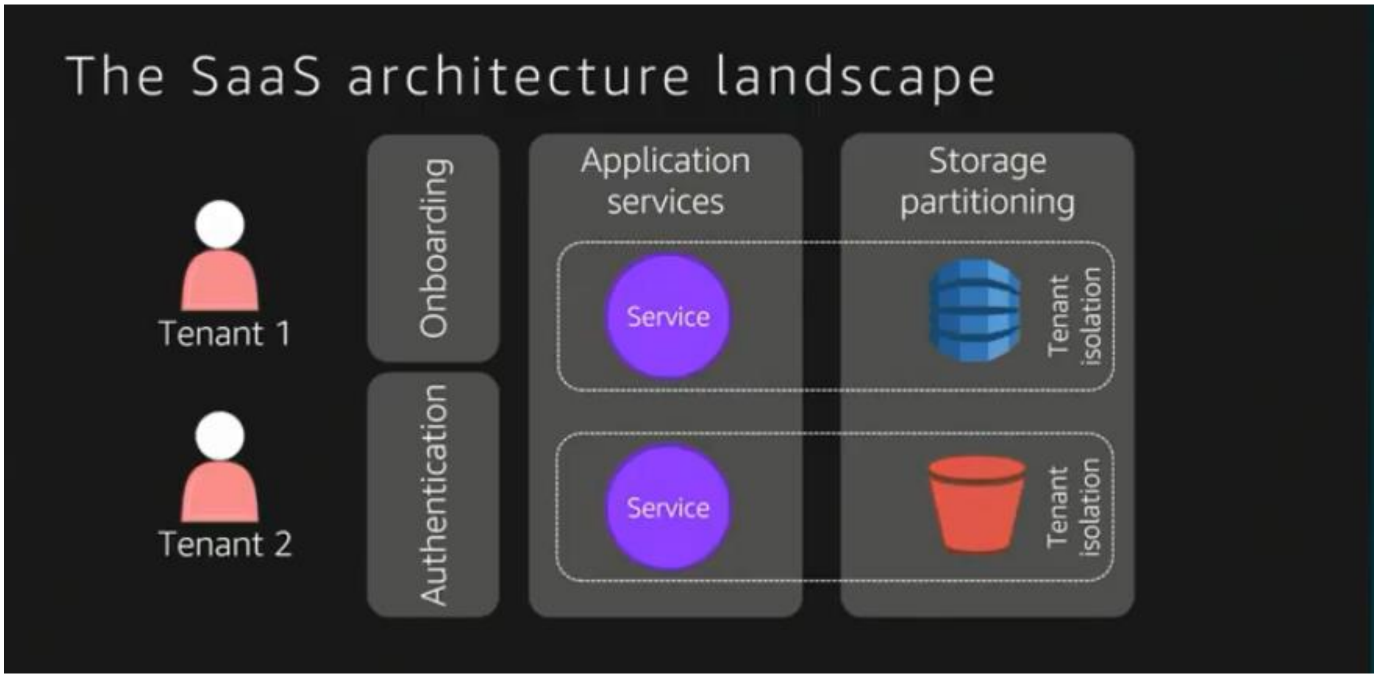
We then need to set up authentication, what does SaaS do to authentication? Identity is a fundamental part of SaaS since it affects the app services, partitioning, etc



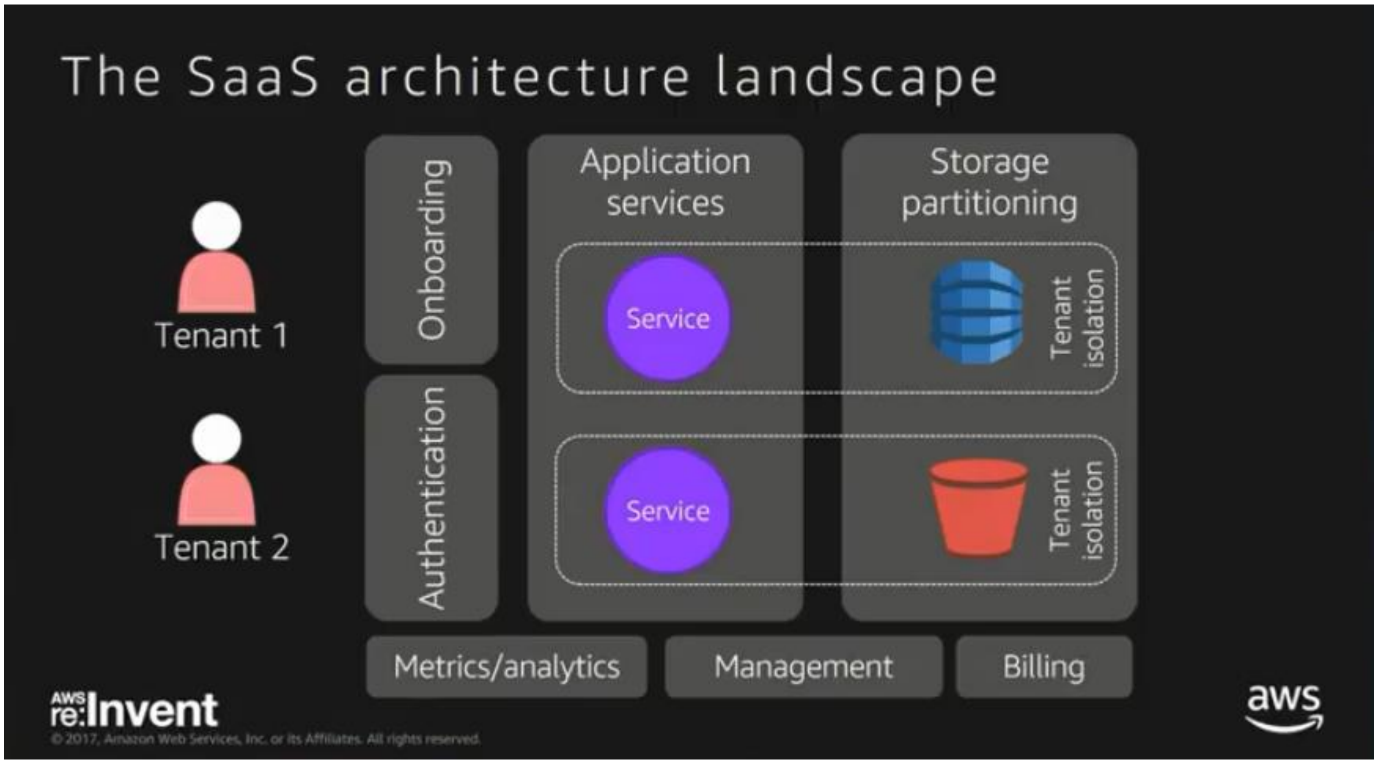
What does it mean to be a developer and write the services as separate microservices in a multi-tenant SaaS environment?



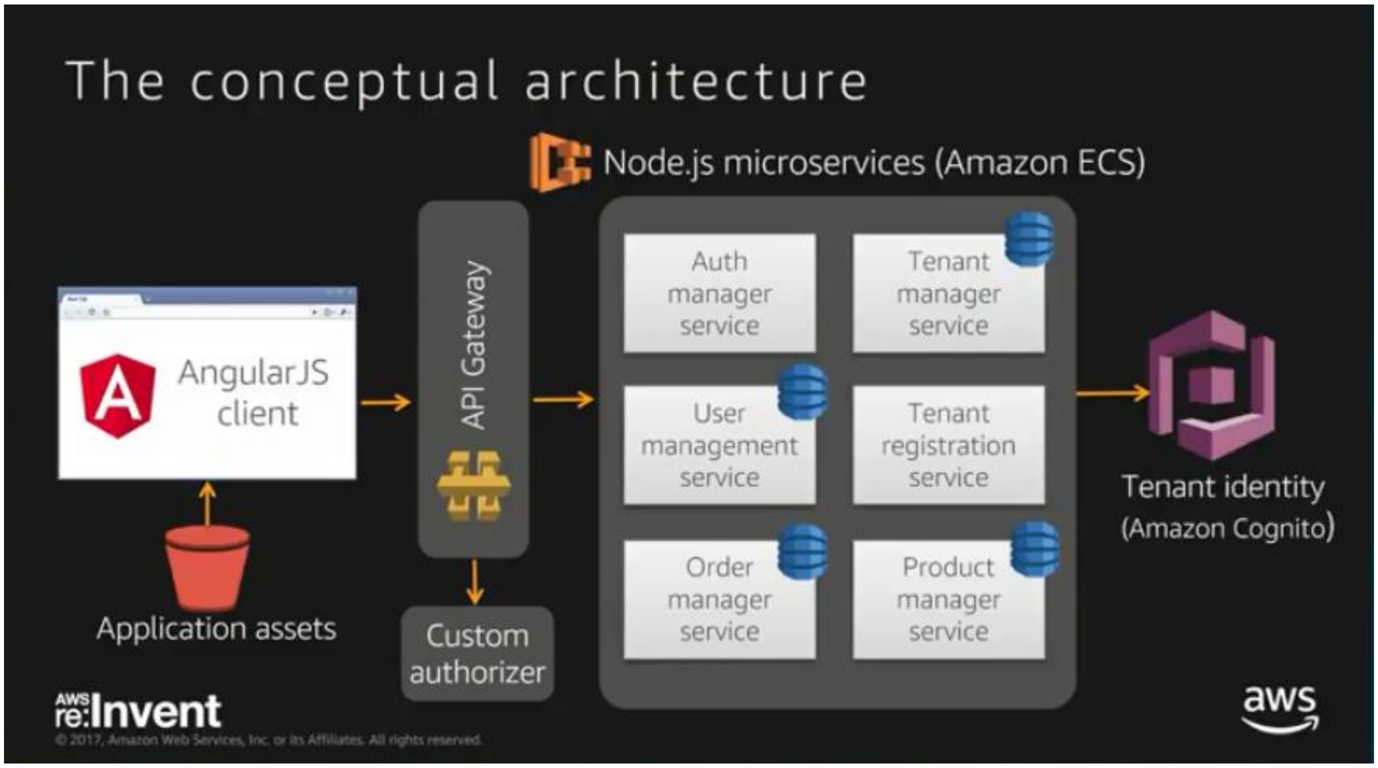
We look at storage partitioning, how we represent pooled, multi-tenant data, how do we encapsulate storage in this SaaS environment



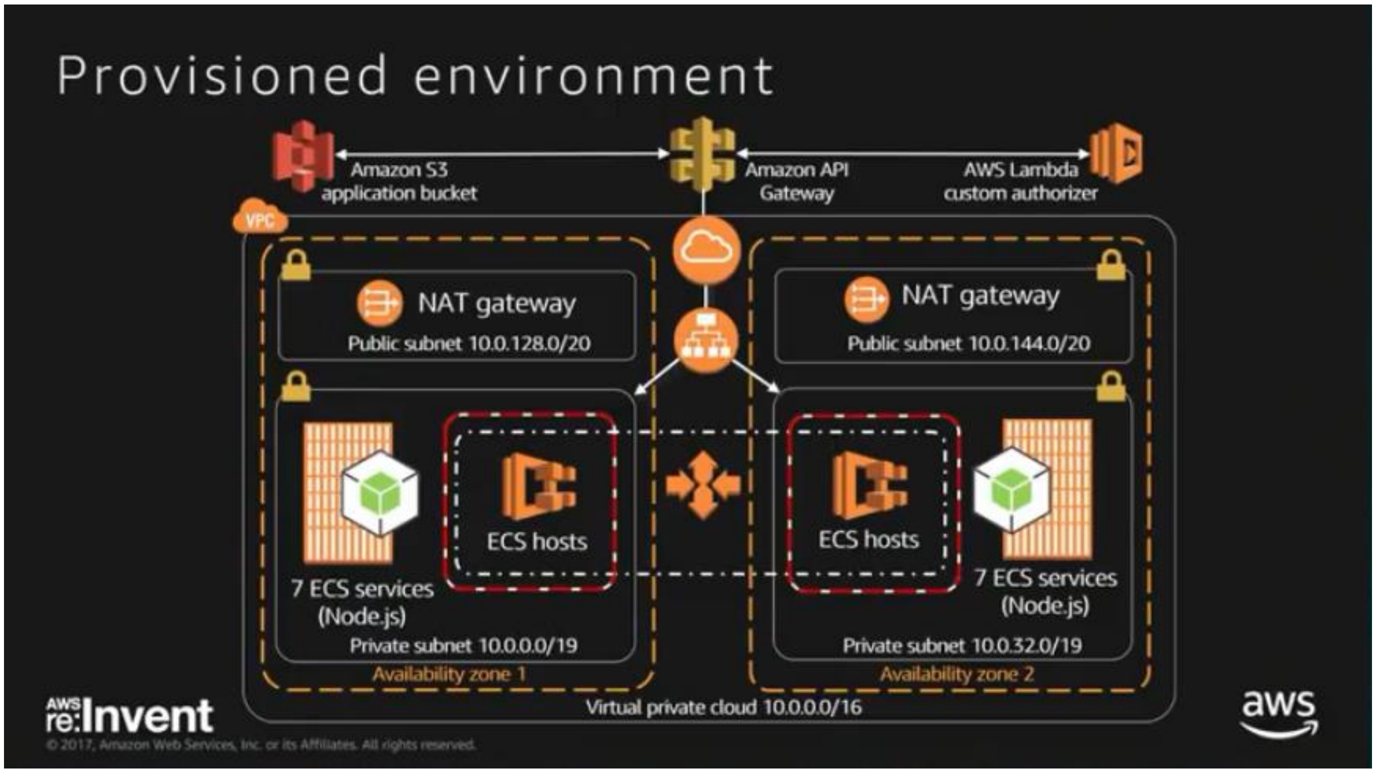
How do we do tenant isolation in a SaaS environment? You really need to keep tight boundaries between tenant resources, what strategies can we have besides authentication to enforce tenant isolation?



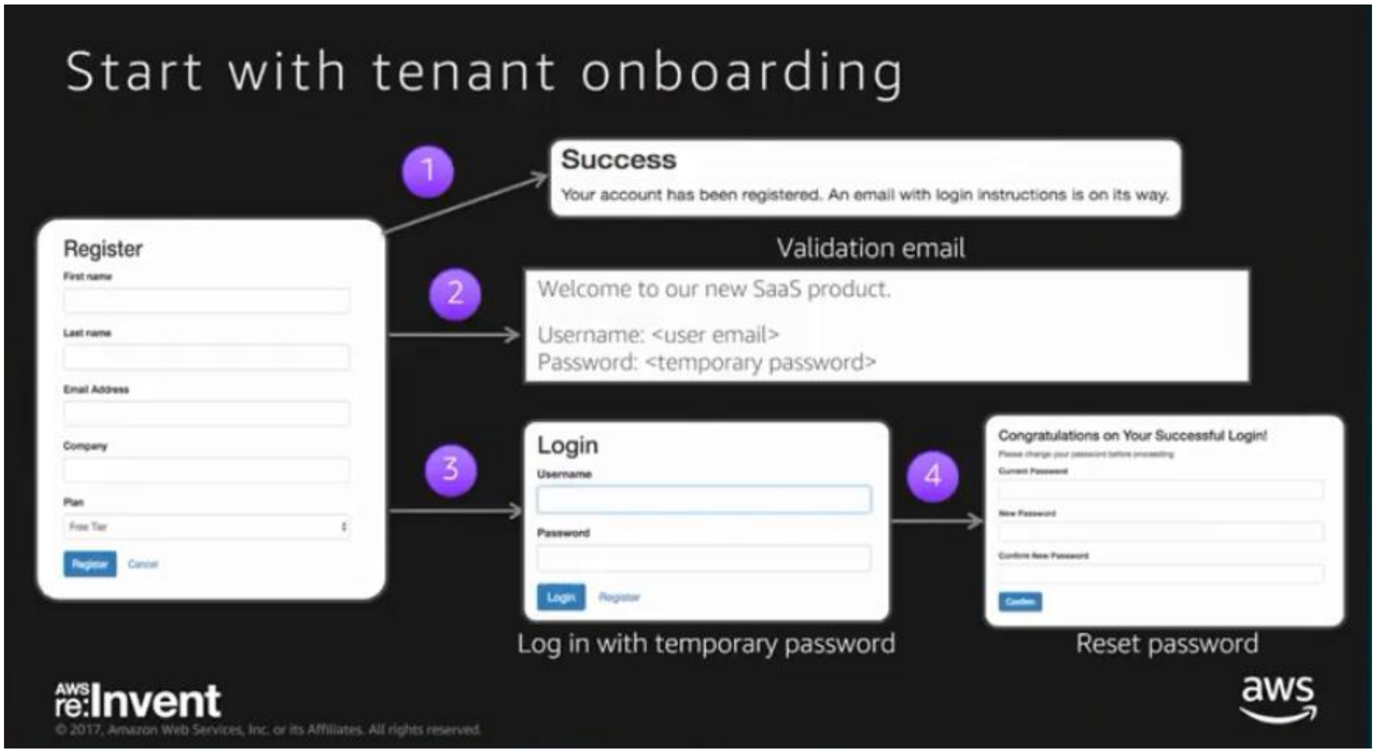
We then have the operational bits of the SaaS applications like health, metrics, management, billing, etc


The conceptual architecture
Node.js microservices (Amazon ECS)
AngularJS client
Application assets
API Gateway
Custom authorizer
Auth manager service
Tenant manager service
User management service
Tenant registration service
Order manager service
Product manager service
Tenant identity (Amazon Cognito)
re:Invent
aws
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

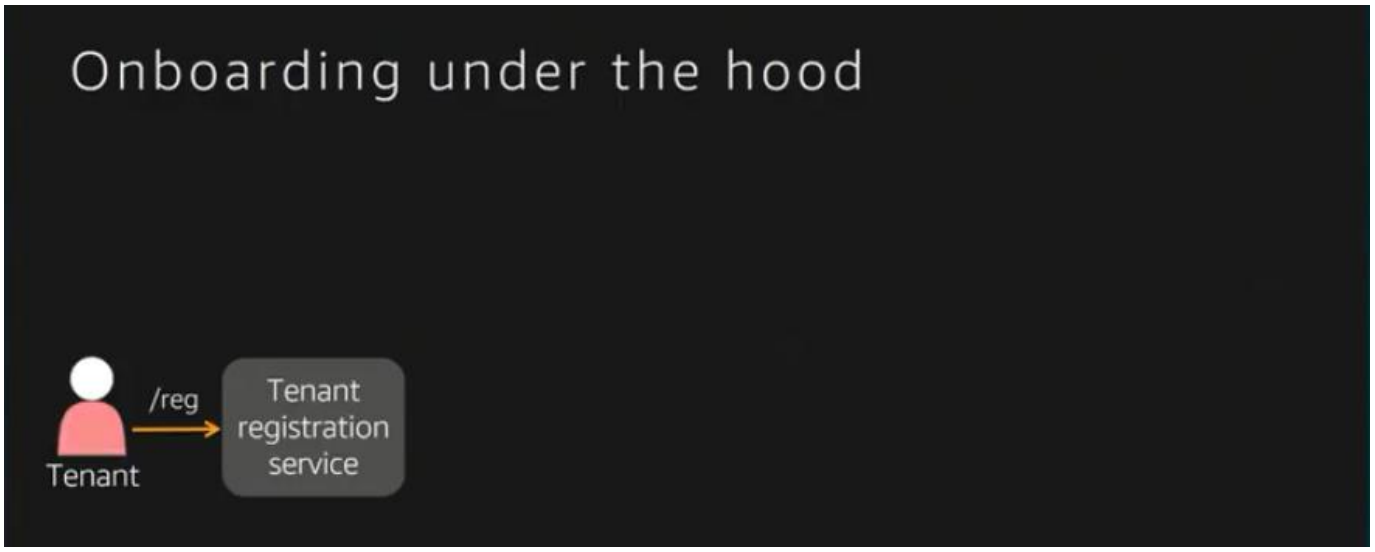
The stack we build this reference SaaS application with is shown above, we use the custom authorizer for an added layer of security, then we have NodeJS services working as separate microservices, we use cognito as the identity management solution and use a lot of its part.


Provisioned environment
Amazon S3 application bucket
Amazon API Gateway
AWS Lambda custom authorizer
VPC
NAT gateway
Public subnet 10.0.128.0/20
NAT gateway
Public subnet 10.0.144.0/20
ECS hosts
ECS hosts
7 ECS services (Node.js)
7 ECS services (Node.js)
Private subnet 10.0.0.0/19
Private subnet 10.0.32.0/19
Availability zone 1
Availability zone 2
Virtual private cloud 10.0.0.0/16
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

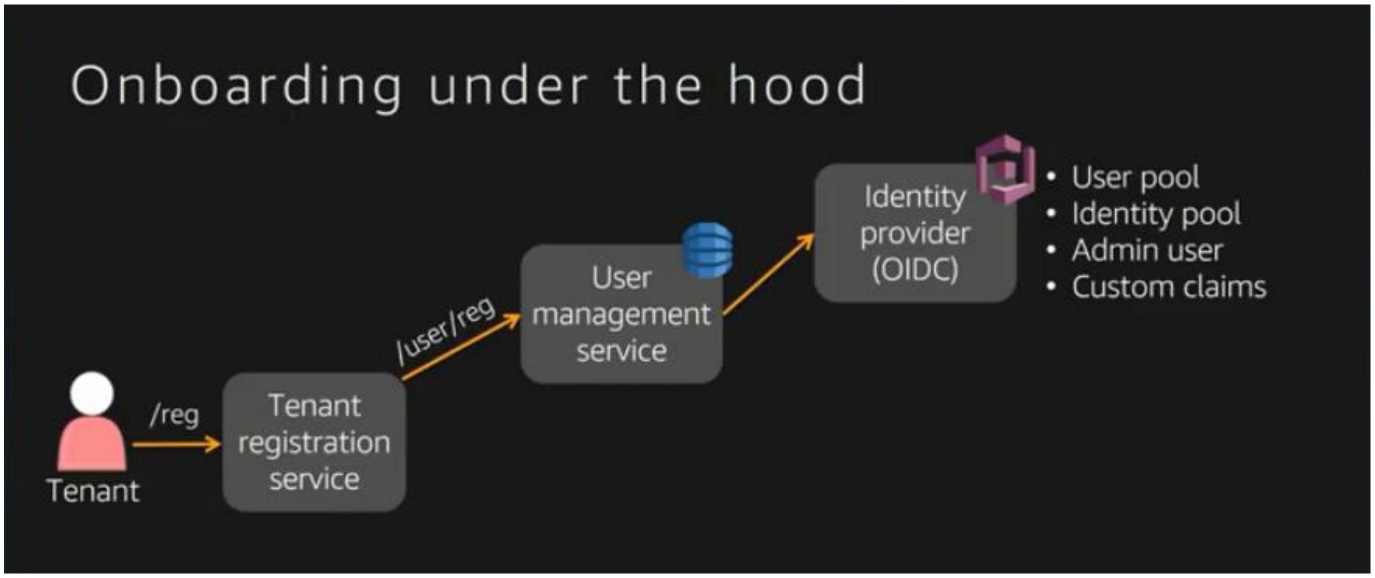
This is the provisioned environment that you will get when you deploy the code, this is an HA architecture


Start with tenant onboarding
1
Success
Your account has been registered. An email with login instructions is on its way.
Register
First name
Last name
Email Address
Company
Plan
Free Tier
Register
Cancel
2
Validation email
Welcome to our new SaaS product.
Username: <user email>
Password: <temporary password>
3
Login
Username
Password
Login
Register
Log in with temporary password
4
Congratulations on Your Successful Login!
Please change your password before proceeding
Current Password
New Password
Confirm New Password
Confirm
Reset password
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

This is the onboarding experience, tell it a little bit about our SaaS configuration, all the code behind this is being done by Cognito for you

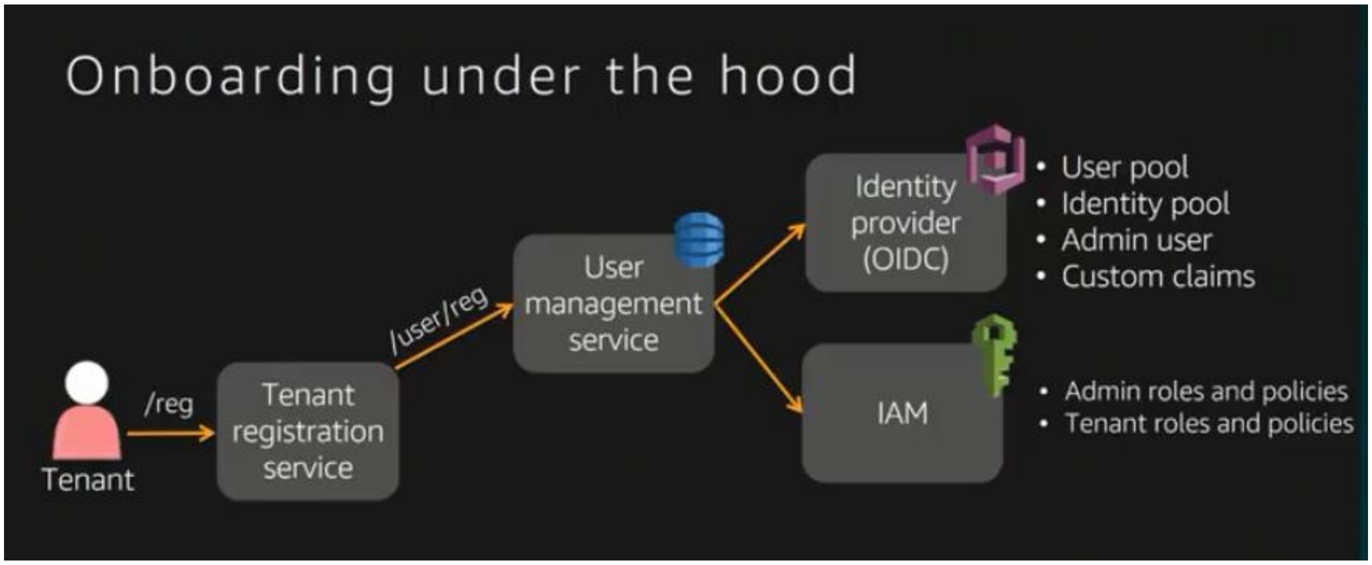

Onboarding under the hood

This Tenant registration service is going to be the orchestrator for all what we have to create for this tenant onboarding after they have successfully registered using Cognito. There are 3 major parts of this service.

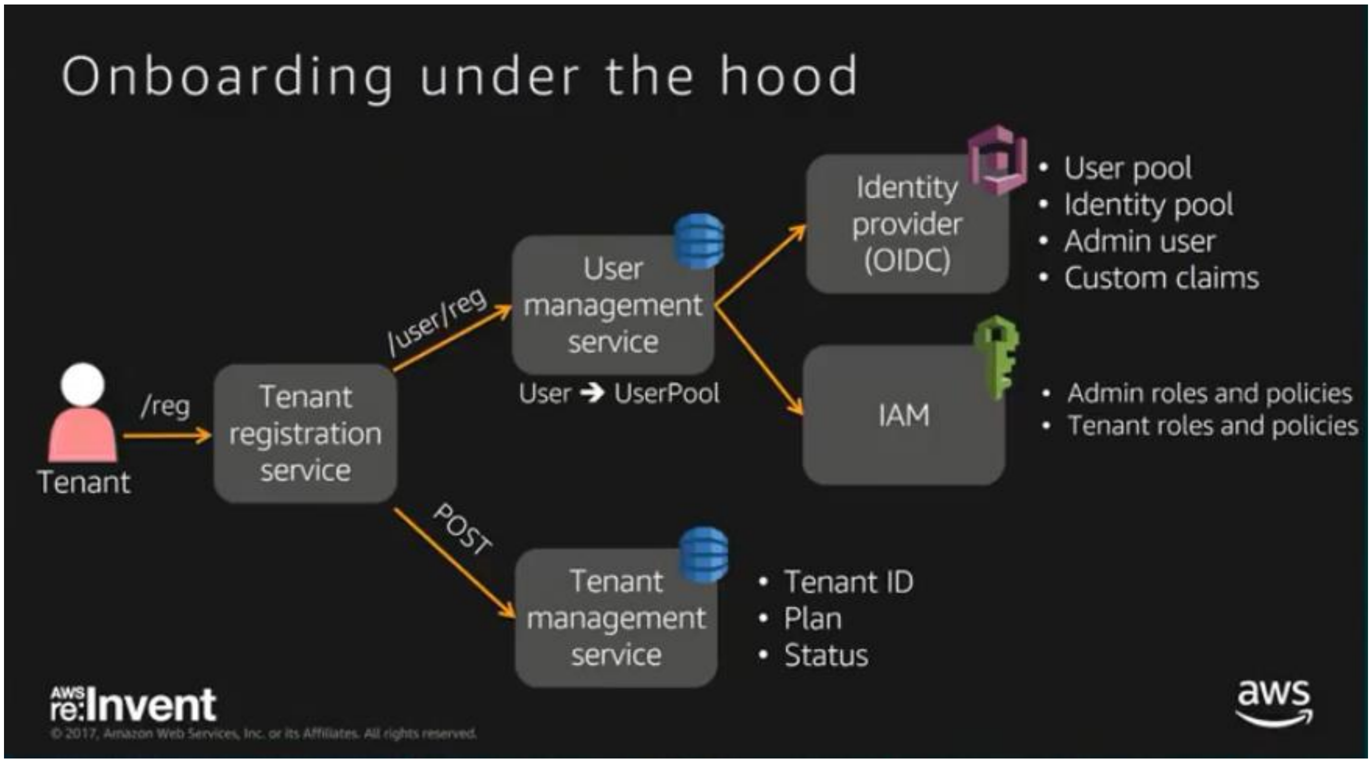

Onboarding under the hood

- User pool
- Identity pool
- Admin user
- Custom claims

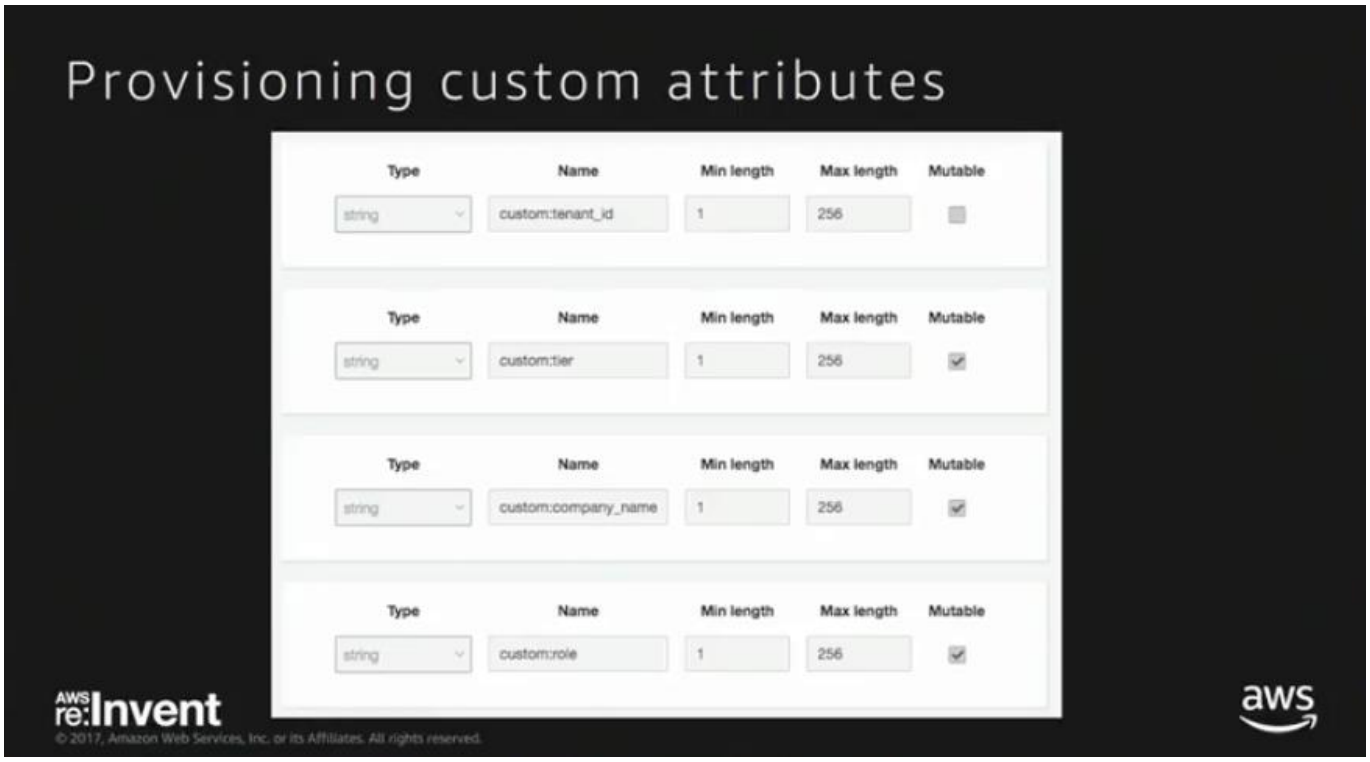
First, we have to create the user and the identity footprint of the app. We have to provision all the mechanics for all the users of the system that will need identity profiles. It uses cognito to create the identity for the user, we are using cognito as an openID connect. It provisions the user pool that lets you configure policies like password policy, MFA. We are provisioning a user pool per tenant.



We also have to provision all the policies that will be needed for all the types of users. We then create some mapping between a user and a user pool.



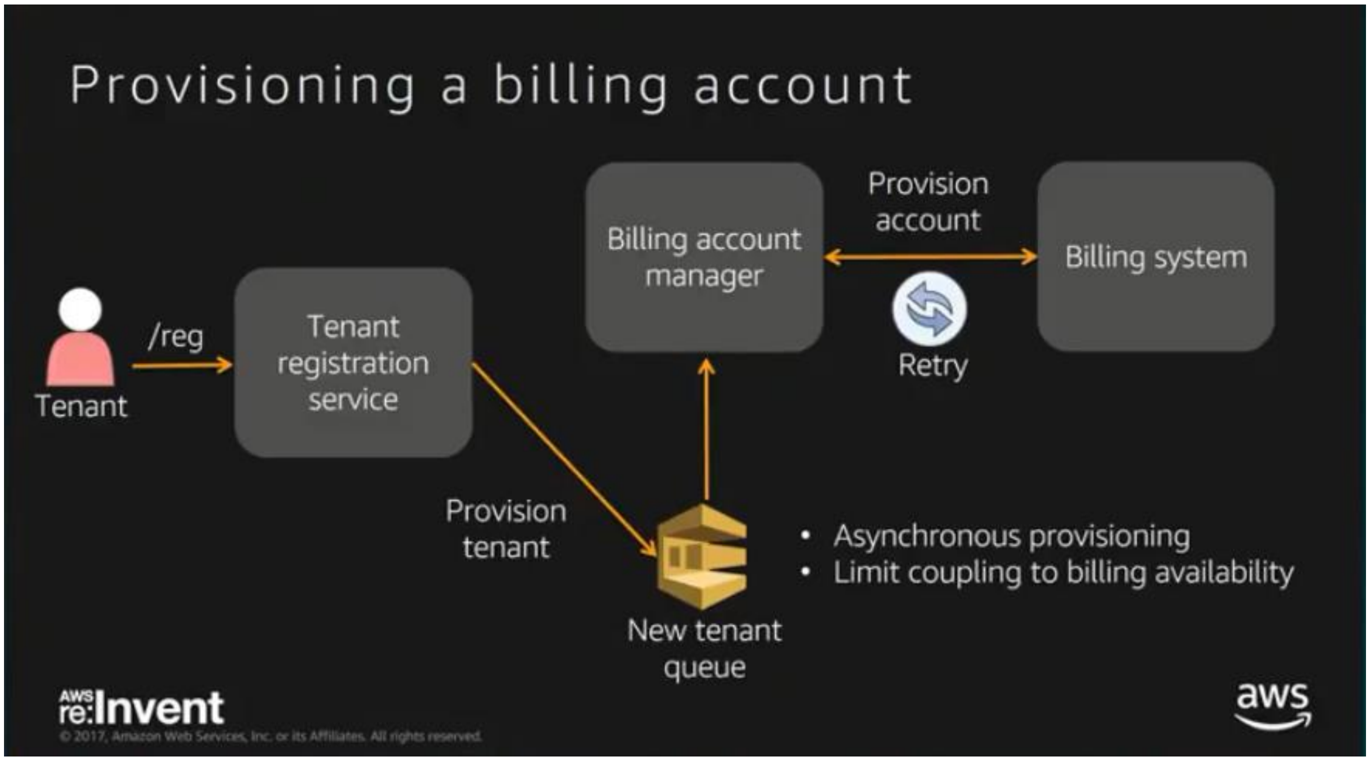
We then create one entry in the DynamoDB table that will have the TenantID, plan, status like active/inactive. Once this is all in place, we will have 1 tenant.



Provisioning custom attributes

Type	Name	Min length	Max length	Mutable
string	custom:tenant_id	1	256	☐
string	custom:tier	1	256	✓
string	custom:company_name	1	256	✓
string	custom:role	1	256	✓

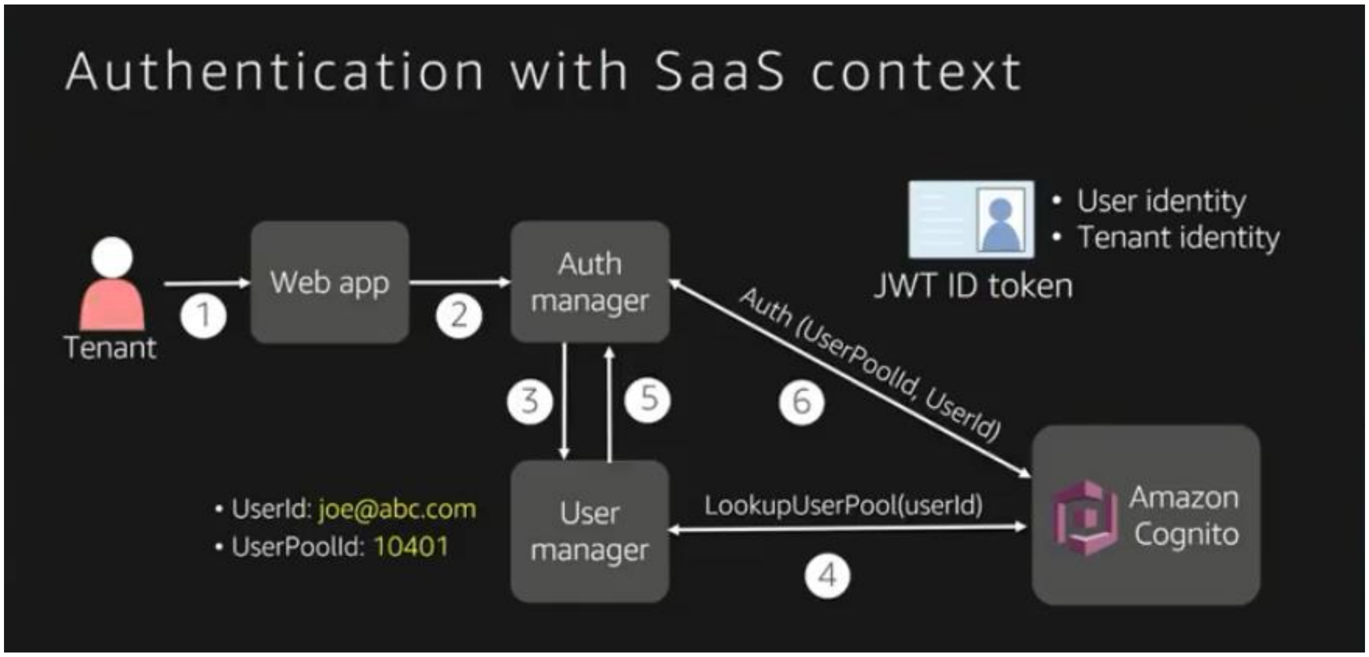
Then we have to set up and configure the tenant itself,



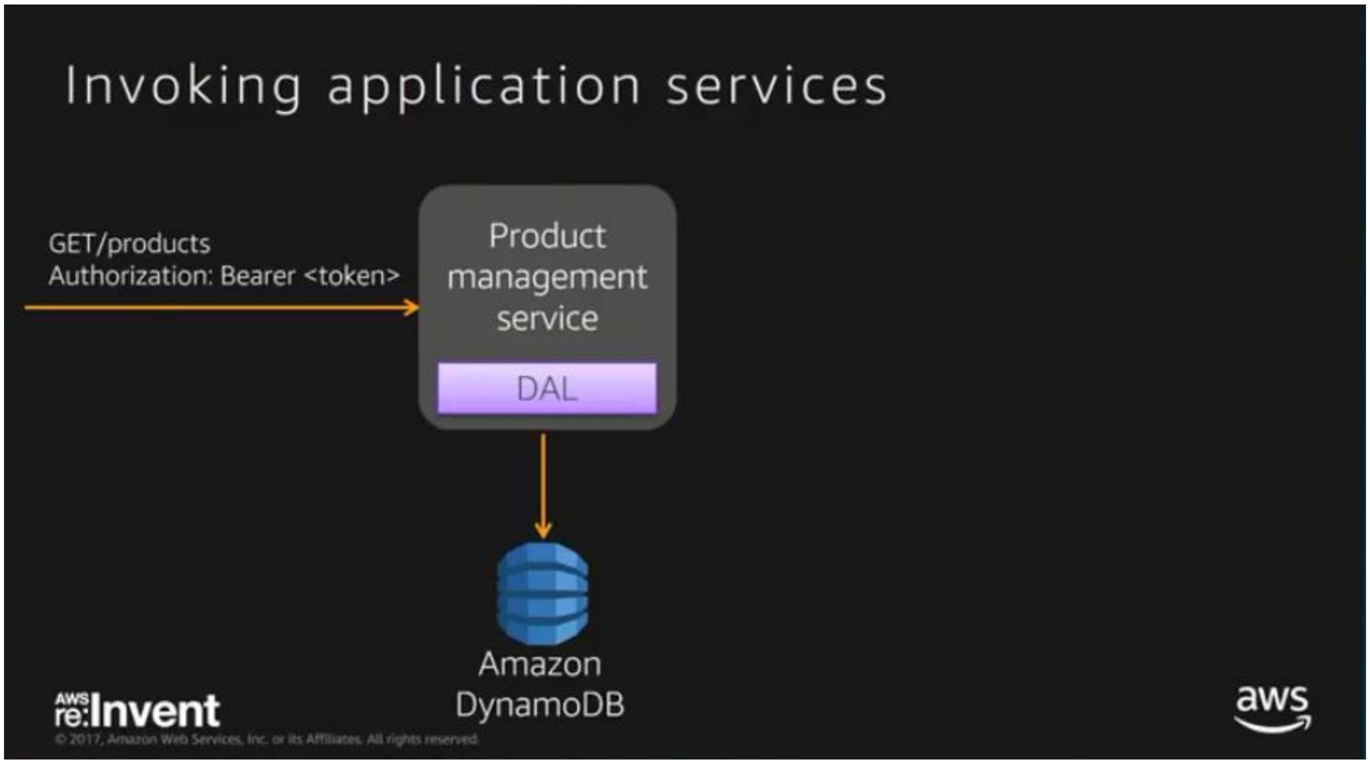
Provisioning a billing account

- Asynchronous provisioning
- Limit coupling to billing availability

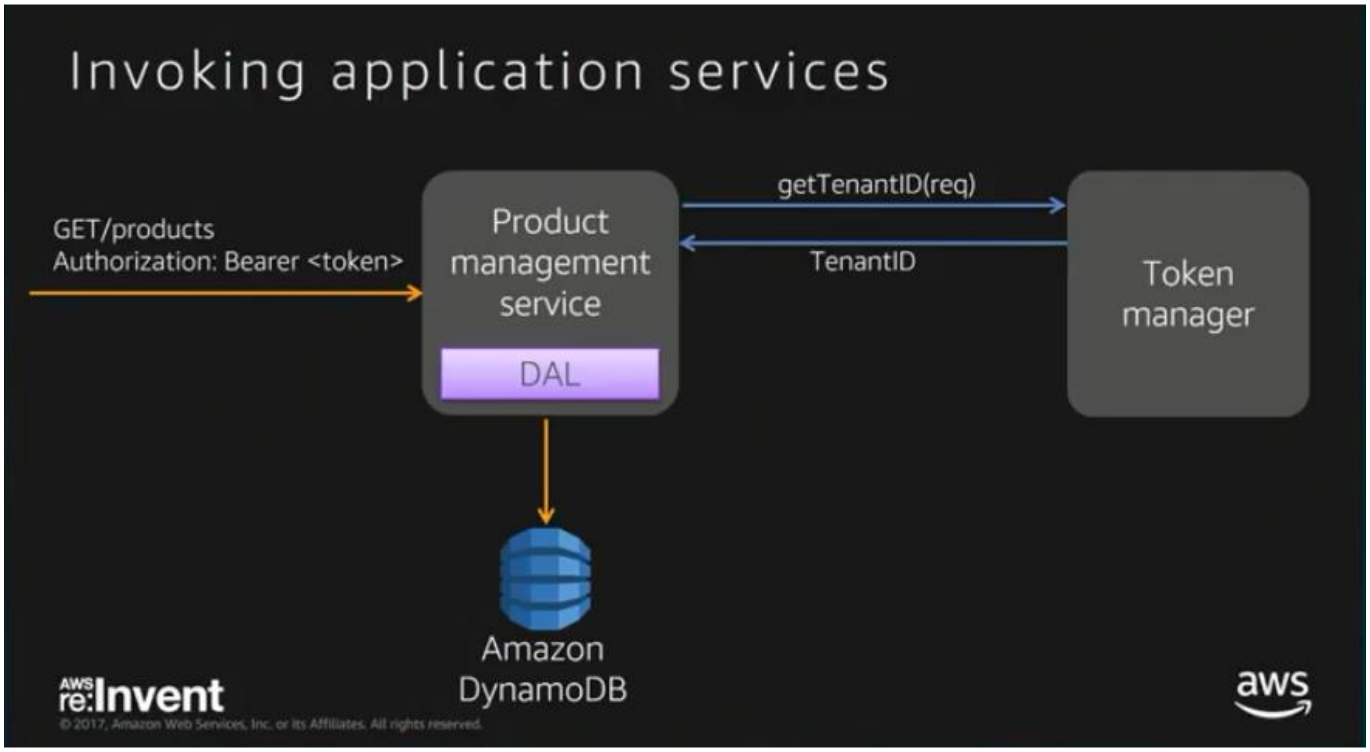
Then we have to set up the actual billing interface for this tenant and provision their billing process and setup.



The encrypted JWT token contains the data we need as a UserID and a Tenant ID that we will use with in the header of all calls to all the downstream SaaS services

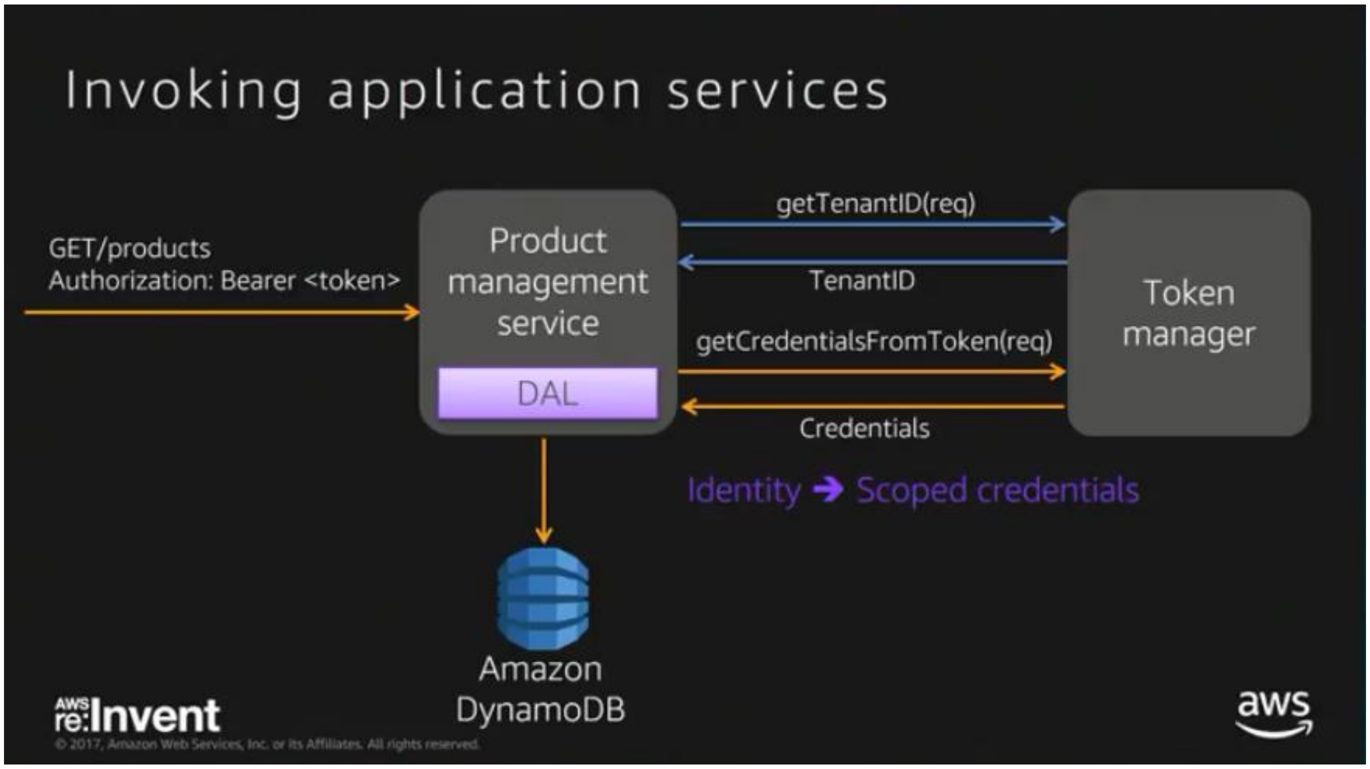

Invoking application services
GET/products
Authorization: Bearer <token>
Product management service
DAL
Amazon DynamoDB
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

We don’t know who you are as a tenant and what IAM credentials to use for access to scope you to within the SaaS services,


Invoking application services

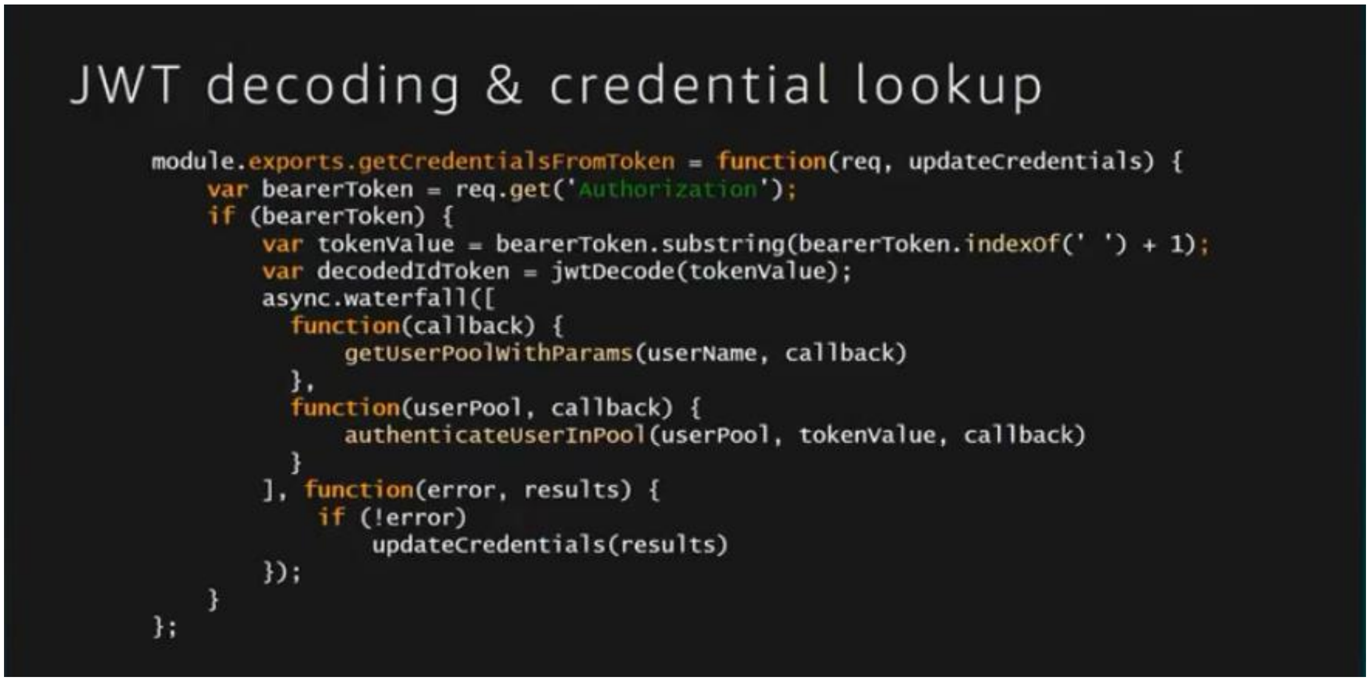
GET/products
Authorization: Bearer <token>
Product management service
DAL
getTenantID(req)
TenantID
Token manager
Amazon DynamoDB
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

We then have a Token manager that gets the Tenant ID out from the JWT token,


Invoking application services

GET/products
Authorization: Bearer <token>
Product management service
DAL
getTenantID(req)
TenantID
getCredentialsFromToken(req)
Credentials
Token manager
Identity ➔ Scoped credentials
Amazon DynamoDB
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

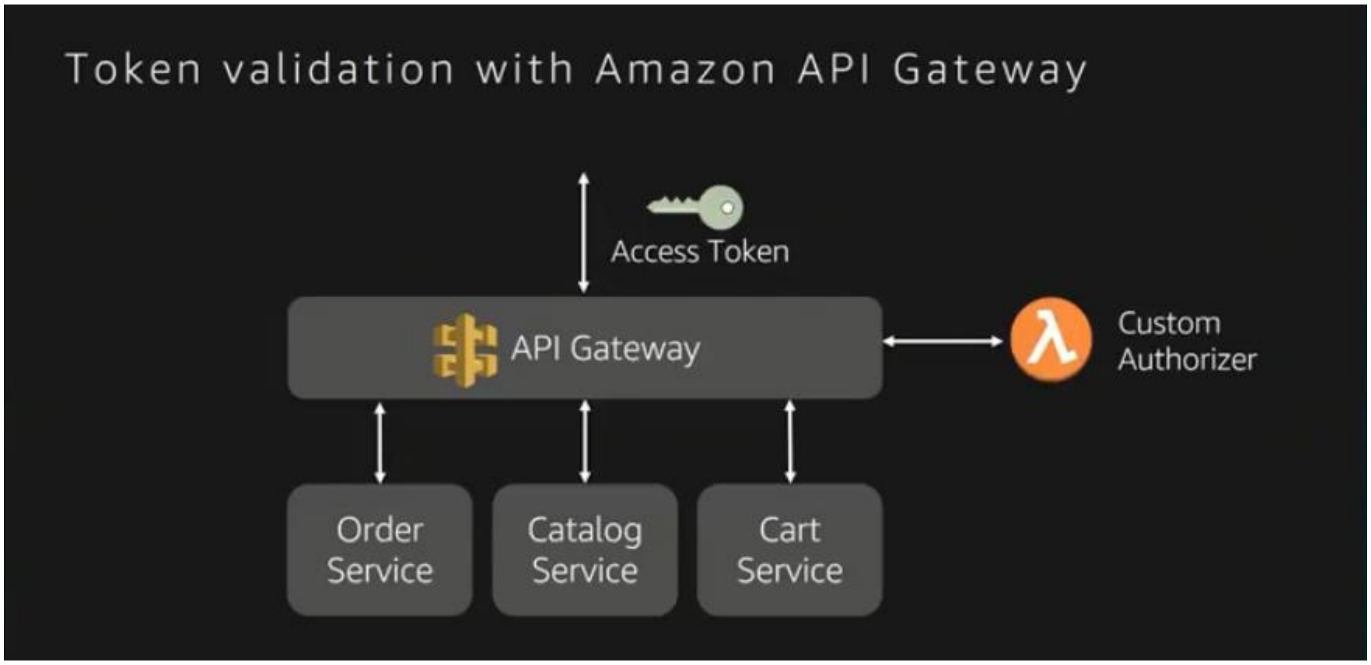
We then get IAM credentials that will give back what the user with the Tenant ID can get/see from the services.



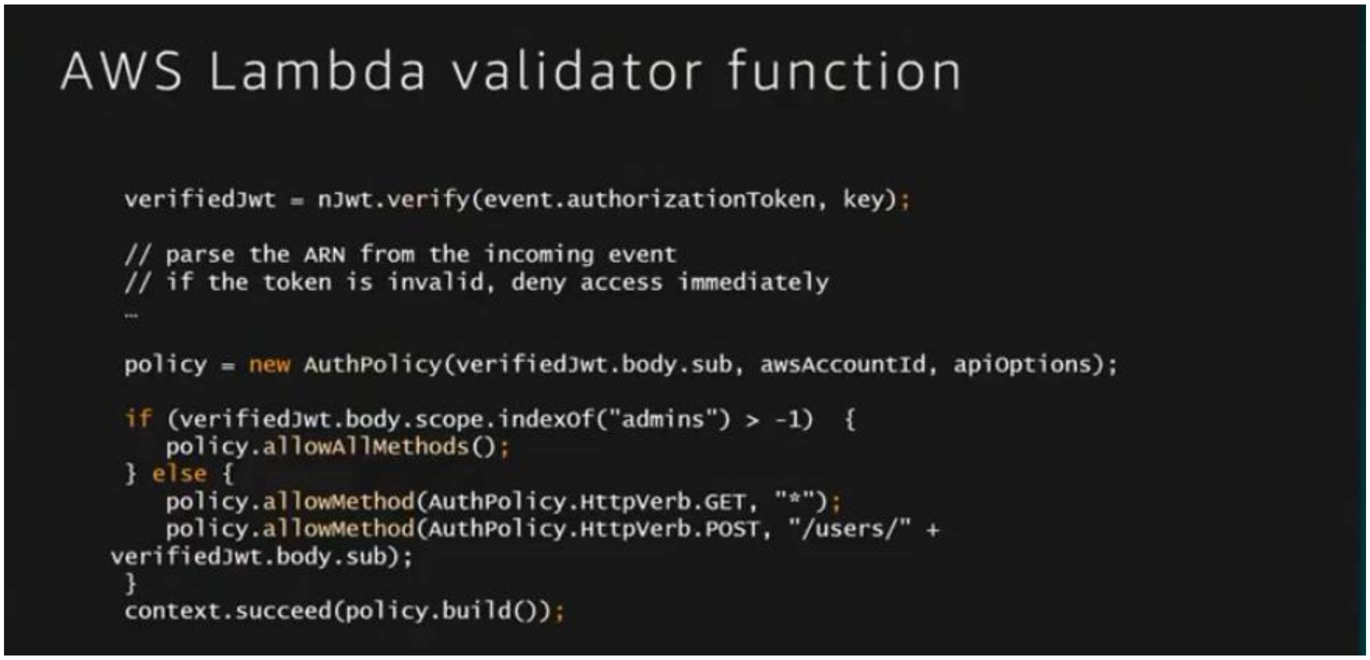
JWT decoding & credential lookup

```
module.exports.getCredentialsFromToken = function(req, updateCredentials) {
    var bearerToken = req.get('Authorization');
    if (bearerToken) {
        var tokenValue = bearerToken.substring(bearerToken.indexOf(' ') + 1);
        var decodedIdToken = jwtDecode(tokenValue);
        async.waterfall([
          function(callback) {
              getUserPoolWithParams(userName, callback)
          },
          function(userPool, callback) {
              authenticateUserInPool(userPool, tokenValue, callback)
          }
        ], function(error, results) {
            if (!error)
                updateCredentials(results)
        });
    }
};
```

Token validation with Amazon API Gateway



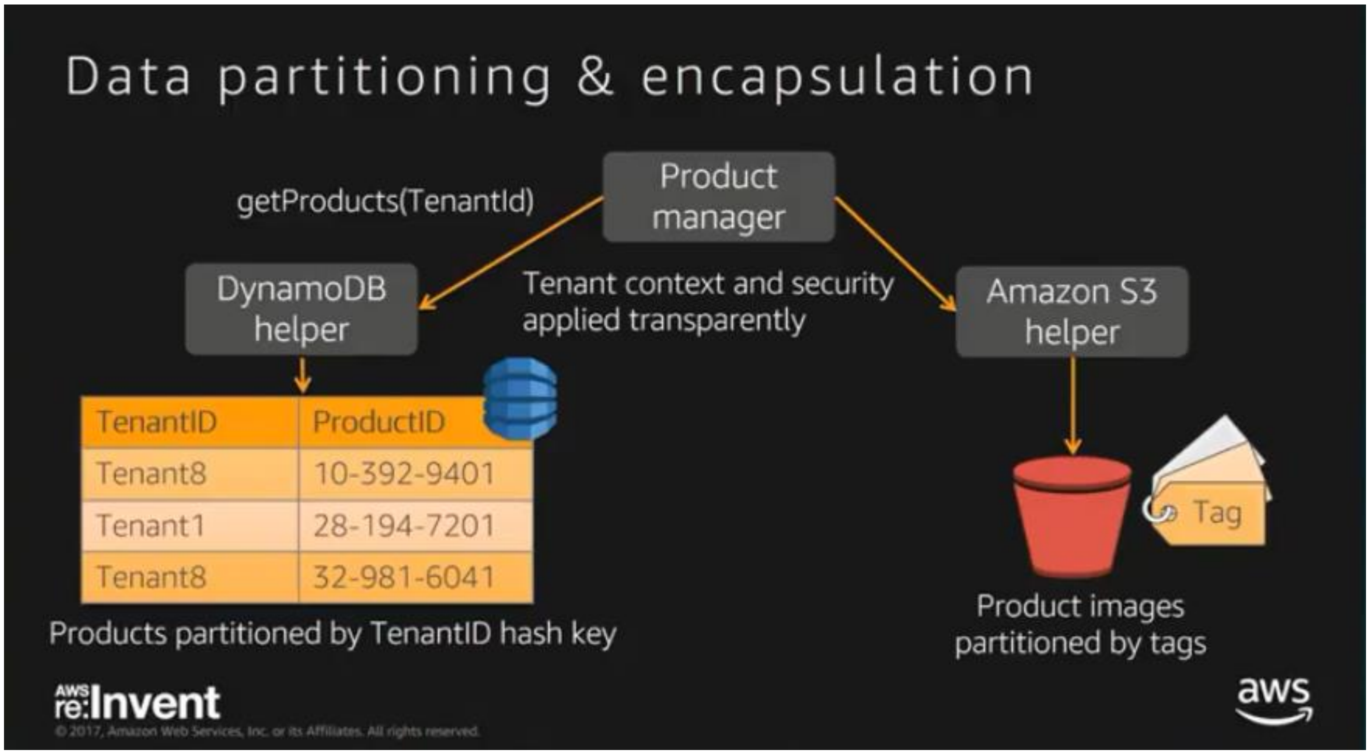
We add a custom authorizer to the API Gateway to add more security by taking the JWT token and inspecting it to determine which API endpoints the user is allowed to call or not



AWS Lambda validator function

```
verifiedJwt = nJwt.verify(event.authorizationToken, key);

// parse the ARN from the incoming event
// if the token is invalid, deny access immediately
…

policy = new AuthPolicy(verifiedJwt.body.sub, awsAccountId, apiOptions);

if (verifiedJwt.body.scope.indexOf("admins") > -1)  {
   policy.allowAllMethods();
} else {
   policy.allowMethod(AuthPolicy.HttpVerb.GET, "*");
   policy.allowMethod(AuthPolicy.HttpVerb.POST, "/users/" +
verifiedJwt.body.sub);
}
context.succeed(policy.build());
```

The policy we create and return is then used by the API Gateway to allow access to an endpoint or not



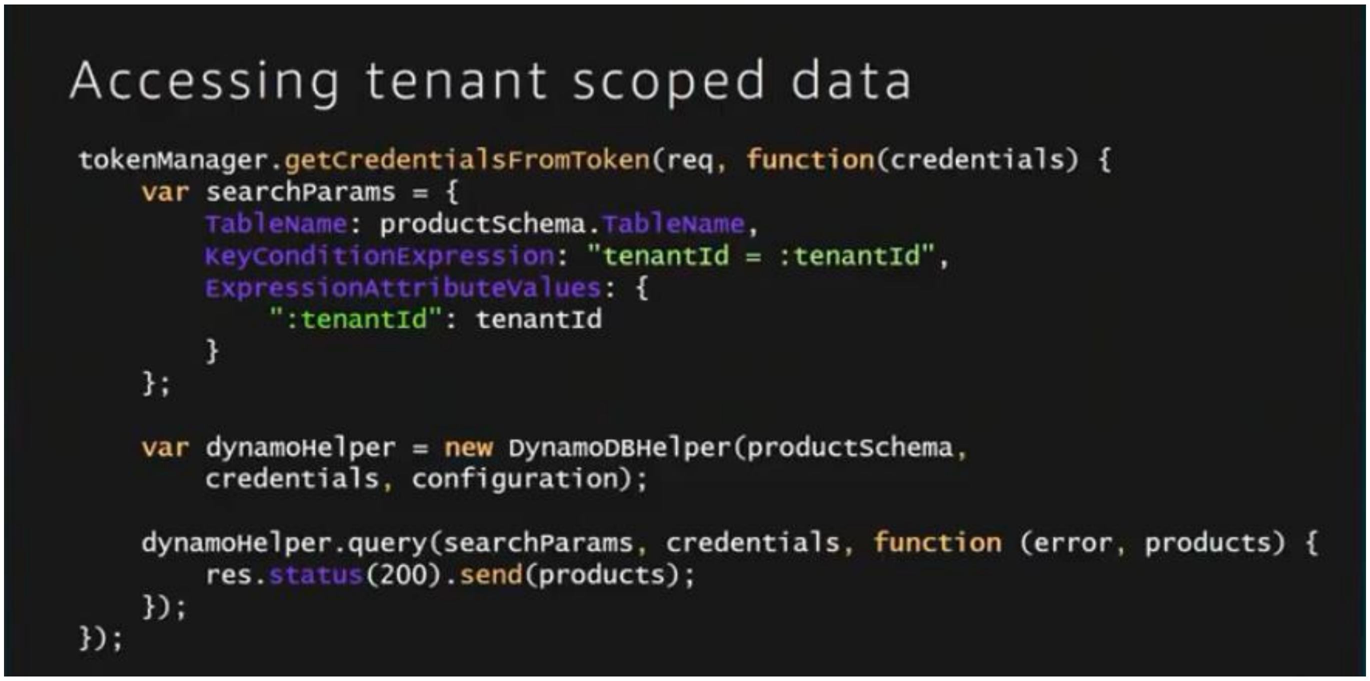
Data partitioning & encapsulation

getProducts(TenantId)
Product manager
DynamoDB helper
Tenant context and security applied transparently
Amazon S3 helper
Tag
Product images partitioned by tags

TenantID	ProductID
Tenant8	10-392-9401
Tenant1	28-194-7201
Tenant8	32-981-6041

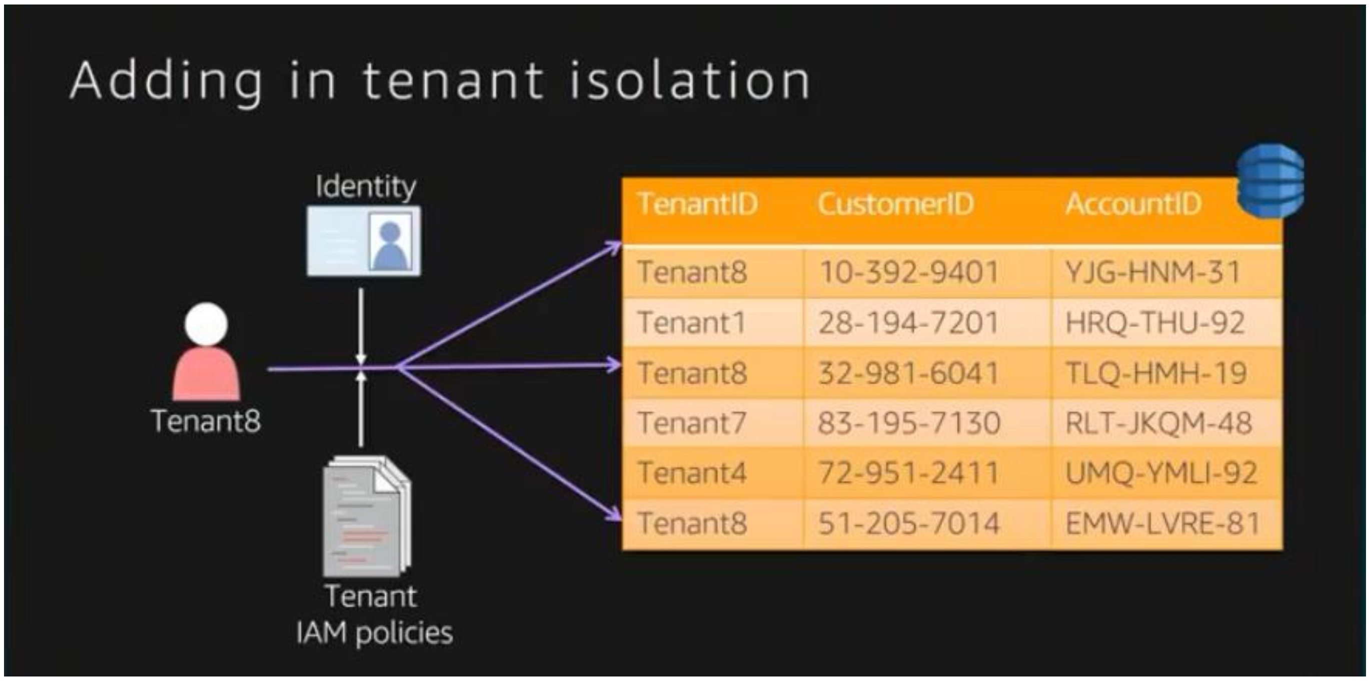
Products partitioned by TenantID hash key

Object tags can also be used to partition data in S3.



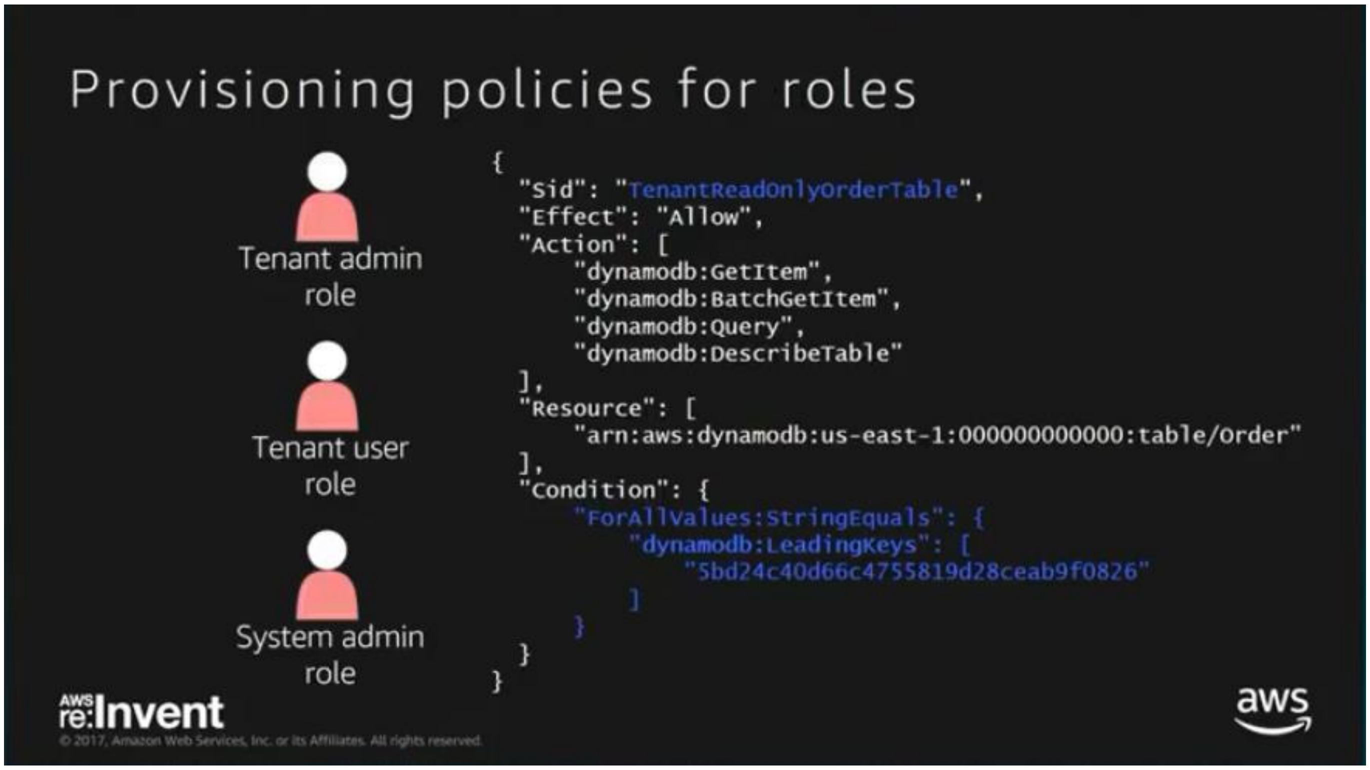
Accessing tenant scoped data

```
tokenManager.getCredentialsFromToken(req, function(credentials) {
    var searchParams = {
        TableName: productSchema.TableName,
        KeyConditionExpression: "tenantId = :tenantId",
        ExpressionAttributeValues: {
            ":tenantId": tenantId
        }
    };

    var dynamoHelper = new DynamoDBHelper(productSchema,
        credentials, configuration);

    dynamoHelper.query(searchParams, credentials, function (error, products) {
        res.status(200).send(products);
    });
});
```



Adding in tenant isolation

TenantID	CustomerID	AccountID
Tenant8	10-392-9401	YJG-HNM-31
Tenant1	28-194-7201	HRQ-THU-92
Tenant8	32-981-6041	TLQ-HMH-19
Tenant7	83-195-7130	RLT-JKQM-48
Tenant4	72-951-2411	UMQ-YMLI-92
Tenant8	51-205-7014	EMW-LVRE-81

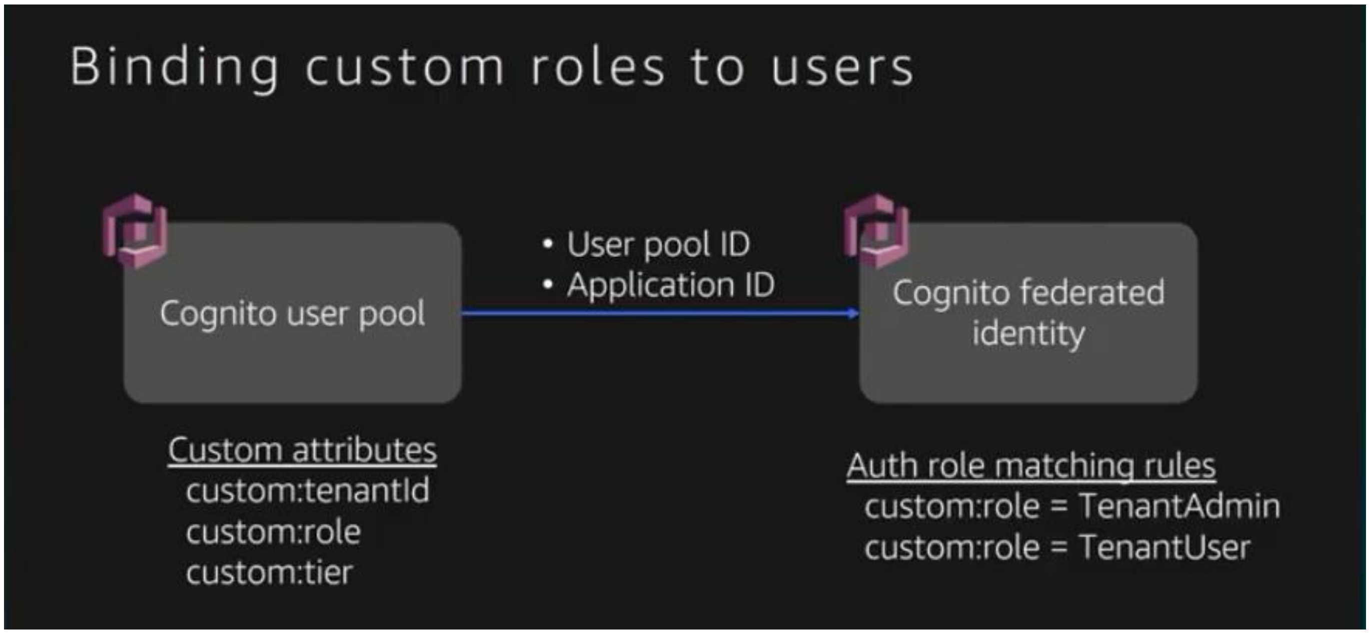


Provisioning policies for roles

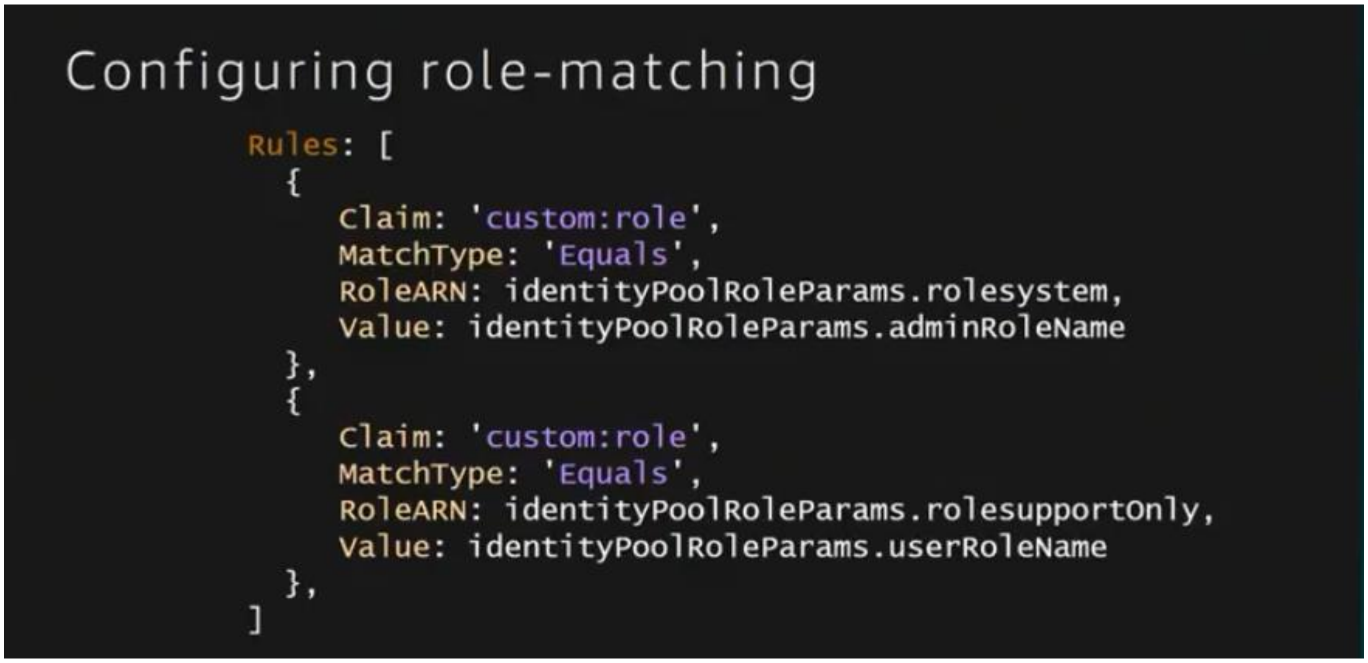
Tenant admin role
Tenant user role
System admin role

```
{
  "Sid": "TenantReadOnlyOrderTable",
  "Effect": "Allow",
  "Action": [
      "dynamodb:GetItem",
      "dynamodb:BatchGetItem",
      "dynamodb:Query",
      "dynamodb:DescribeTable"
  ],
  "Resource": [
      "arn:aws:dynamodb:us-east-1:000000000000:table/Order"
  ],
  "Condition": {
      "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": [
              "5bd24c40d66c4755819d28ceab9f0826"
          ]
      }
  }
}
```

This is one of the IAM policies that were provisioned for a tenant when they register, the LeadingKeys value contains the tenant ID that we want to use this specific policy

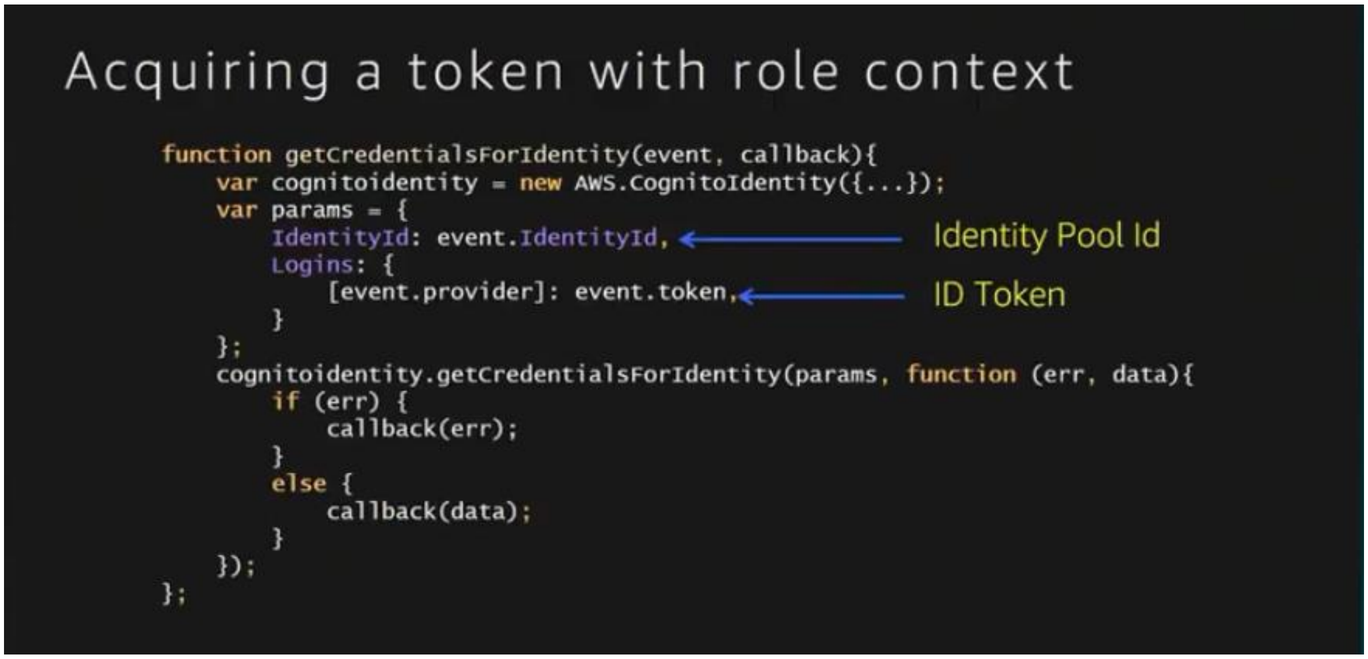


Binding custom roles to users



Configuring role-matching

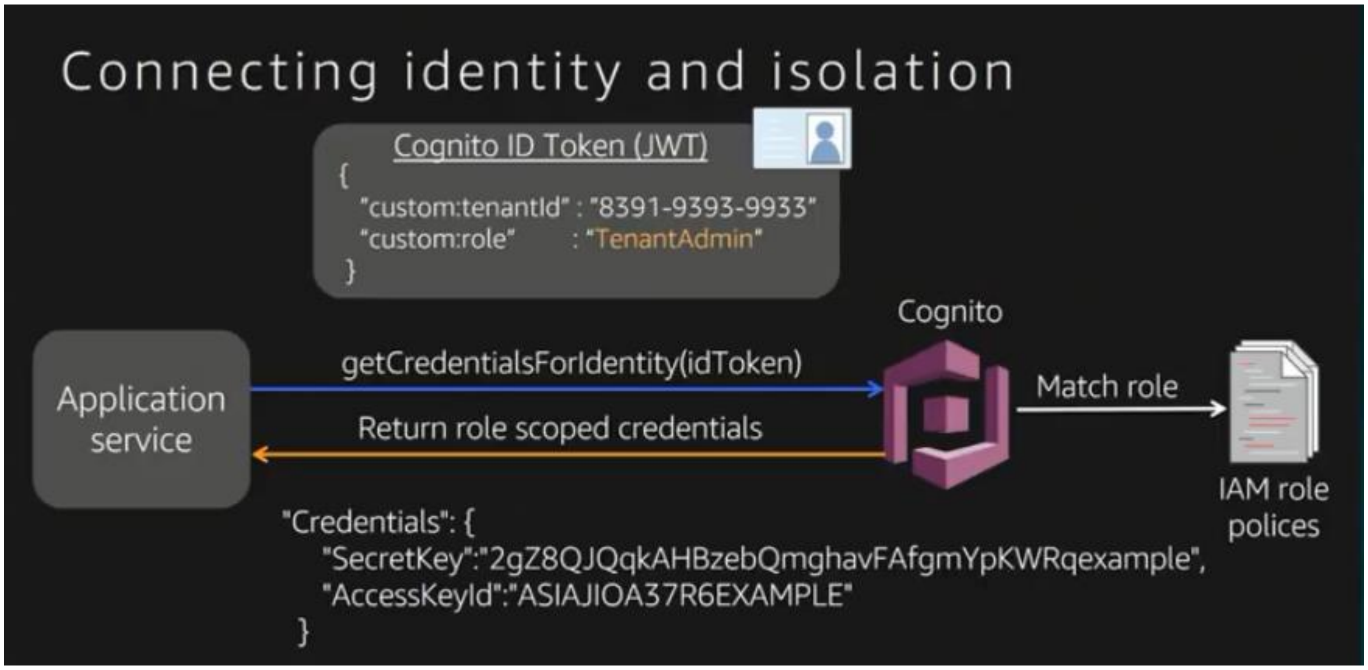
```
Rules: [
  {
     Claim: 'custom:role',
     MatchType: 'Equals',
     RoleARN: identityPoolRoleParams.rolesystem,
     Value: identityPoolRoleParams.adminRoleName
  },
  {
     Claim: 'custom:role',
     MatchType: 'Equals',
     RoleARN: identityPoolRoleParams.rolesupportOnly,
     Value: identityPoolRoleParams.userRoleName
  },
]
```



Acquiring a token with role context

```
function getCredentialsForIdentity(event, callback){
    var cognitoidentity = new AWS.CognitoIdentity({...});
    var params = {
        IdentityId: event.IdentityId,
        Logins: {
            [event.provider]: event.token,
        }
    };
    cognitoidentity.getCredentialsForIdentity(params, function (err, data){
        if (err) {
            callback(err);
        }
        else {
            callback(data);
        }
    });
};
```

Identity Pool Id
ID Token

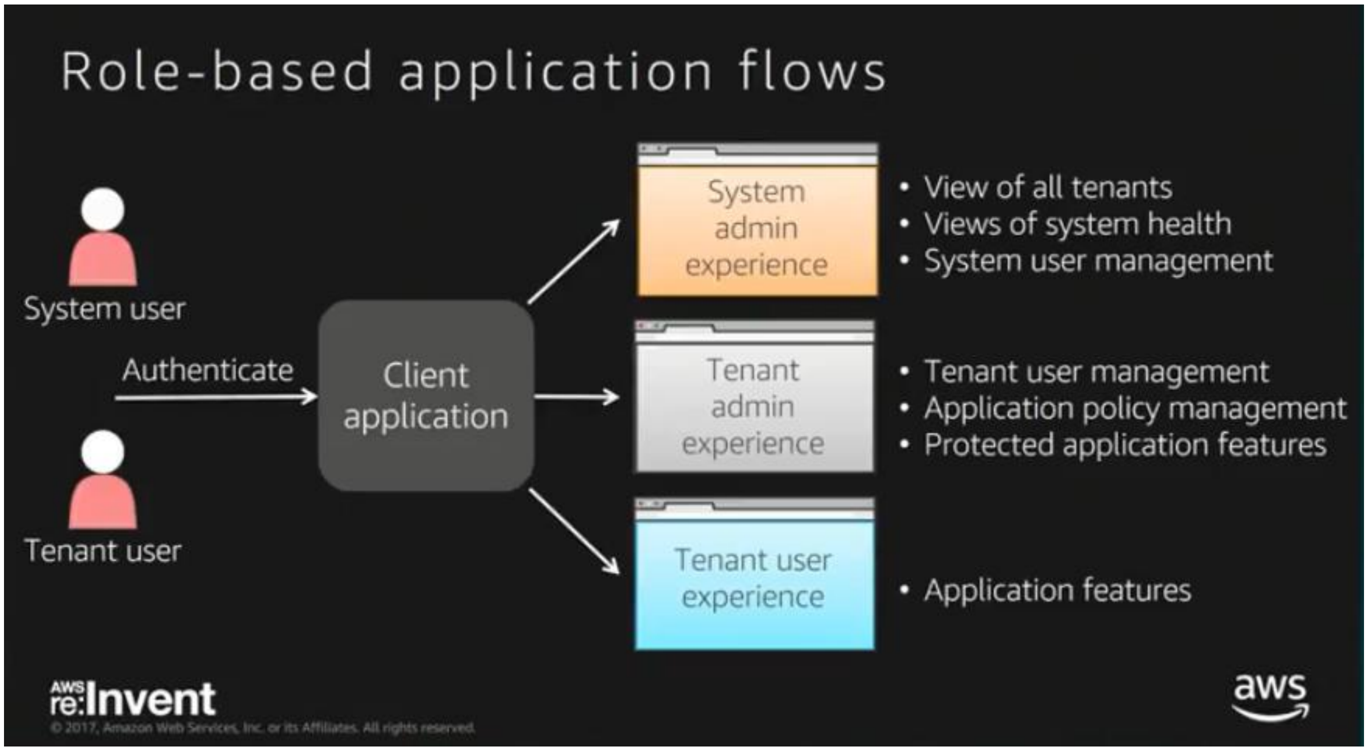


Connecting identity and isolation

```
{
  "custom:tenantId" : "8391-9393-9933"
  "custom:role"     : "TenantAdmin"
}
```

Application service
getCredentialsForIdentity(idToken)
Return role scoped credentials
Cognito
Match role
IAM role polices

```
"Credentials": {
    "SecretKey":"2gZ8QJQqkAHBzebQmghavFAfgmYpKWRqexample",
    "AccessKeyId":"ASIAJIOA37R6EXAMPLE"
 }
```

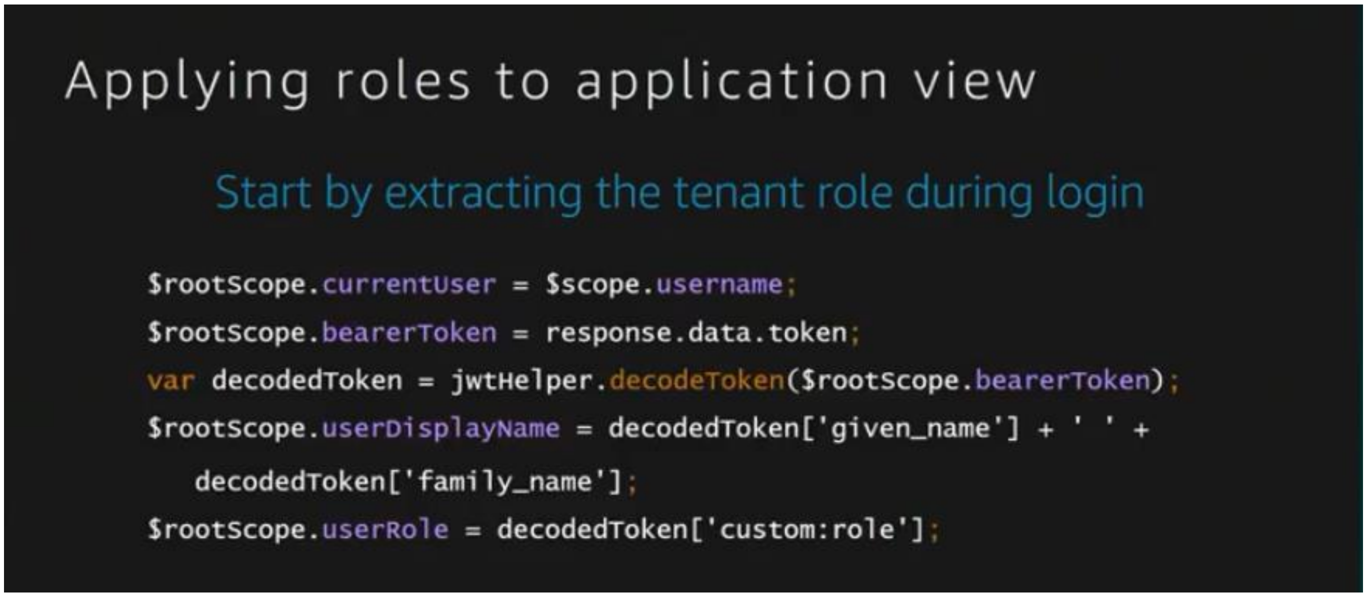


Role-based application flows

- View of all tenants
- Views of system health
- System user management

- Tenant user management
- Application policy management
- Protected application features

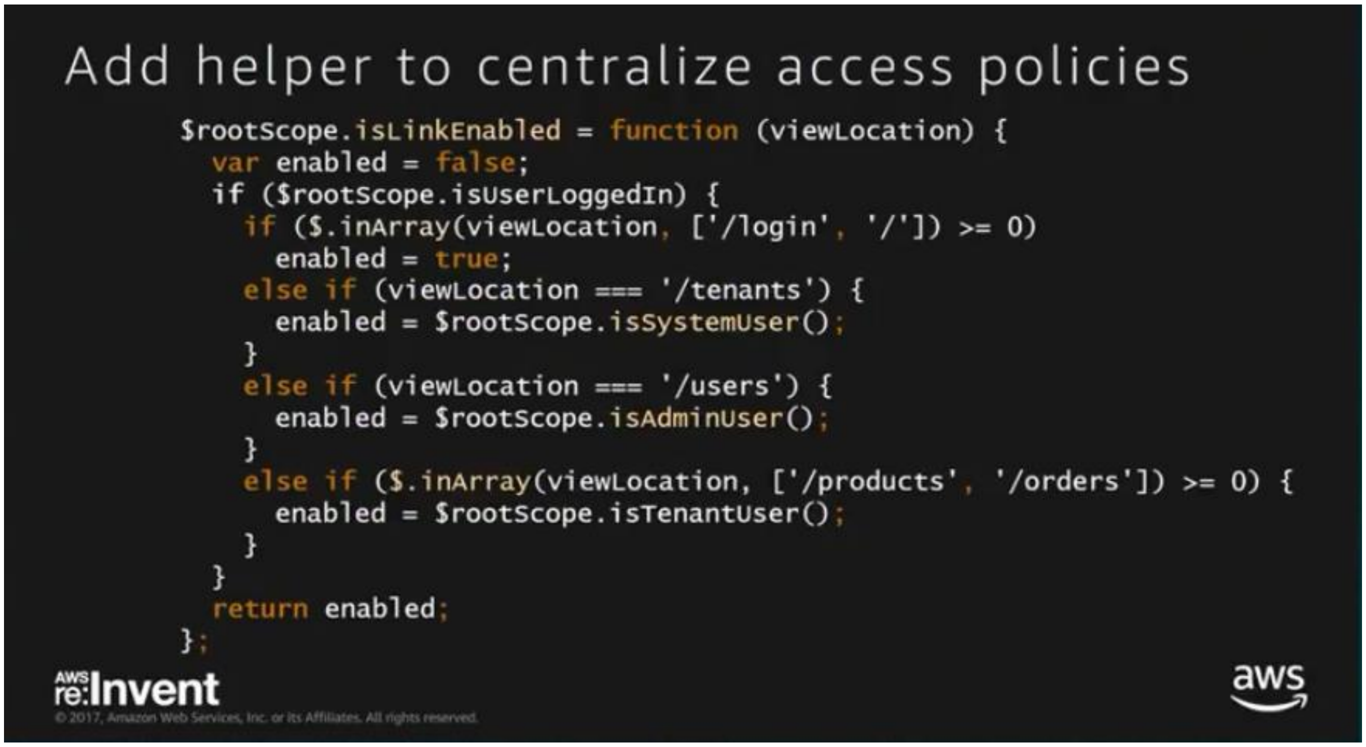
- Application features



Applying roles to application view

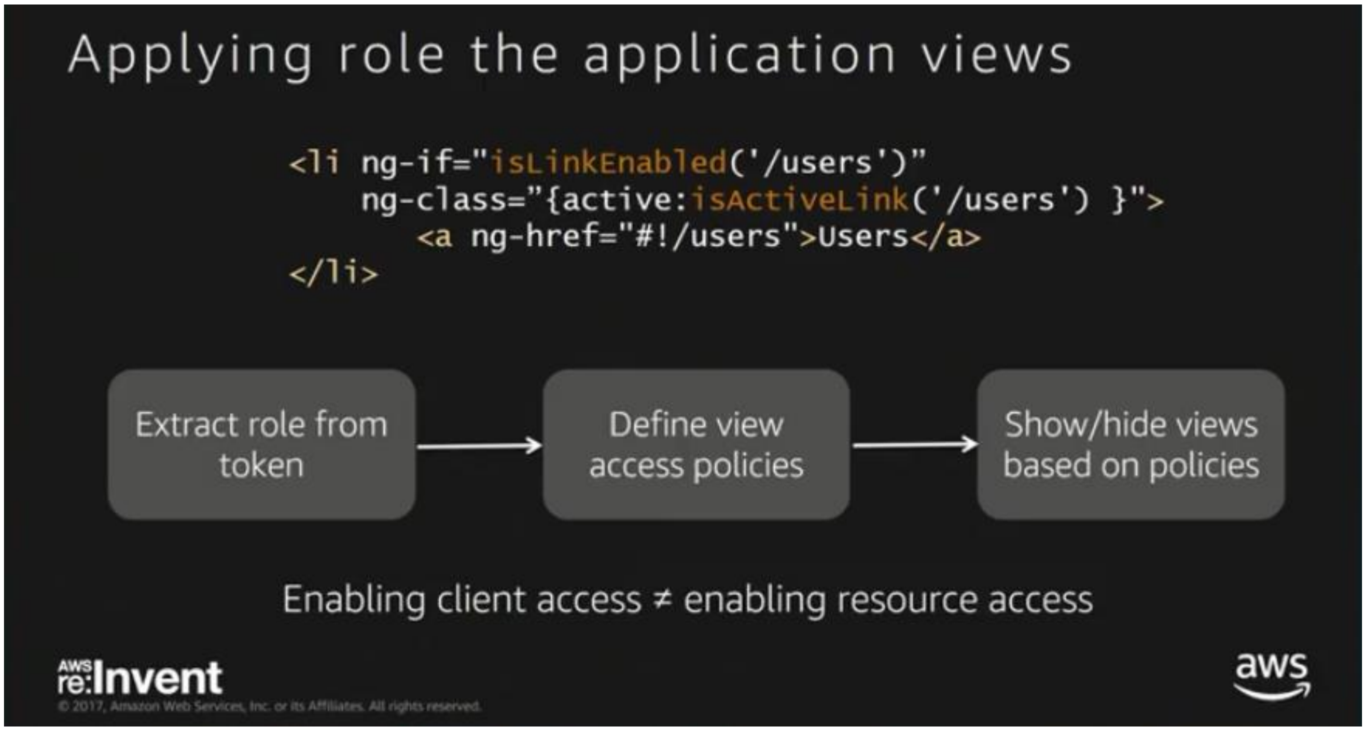
Start by extracting the tenant role during login

```
$rootScope.currentUser = $scope.username;
$rootScope.bearerToken = response.data.token;
var decodedToken = jwtHelper.decodeToken($rootScope.bearerToken);
$rootScope.userDisplayName = decodedToken['given_name'] + ' ' +
   decodedToken['family_name'];
$rootScope.userRole = decodedToken['custom:role'];
```



Add helper to centralize access policies

```
$rootScope.isLinkEnabled = function (viewLocation) {
  var enabled = false;
  if ($rootScope.isUserLoggedIn) {
    if ($.inArray(viewLocation, ['/login', '/']) >= 0)
      enabled = true;
    else if (viewLocation === '/tenants') {
      enabled = $rootScope.isSystemUser();
    }
    else if (viewLocation === '/users') {
      enabled = $rootScope.isAdminUser();
    }
    else if ($.inArray(viewLocation, ['/products', '/orders']) >= 0) {
      enabled = $rootScope.isTenantUser();
    }
  }
  return enabled;
};
```

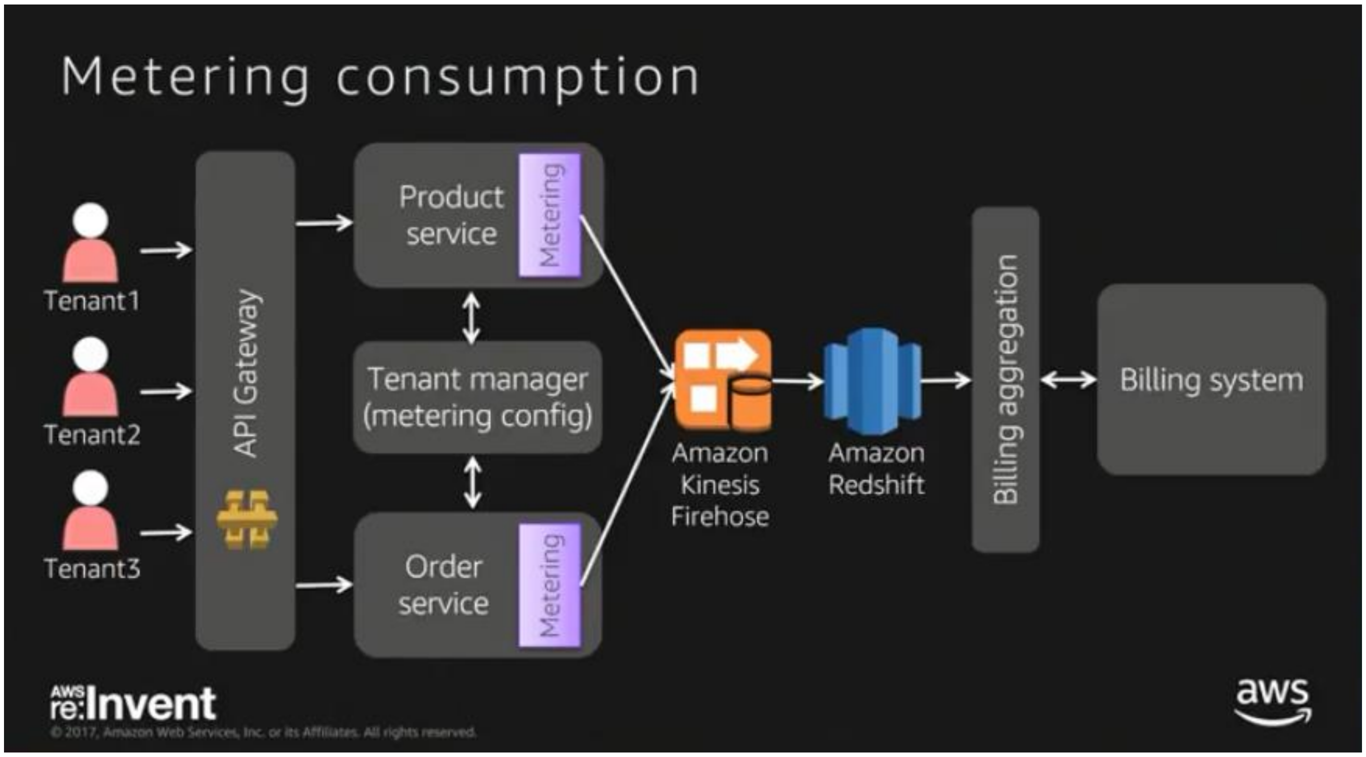


Applying role the application views

```
<li ng-if="isLinkEnabled('/users')"
    ng-class="{active:isActiveLink('/users') }">
        <a ng-href="#!/users">Users</a>
</li>
```

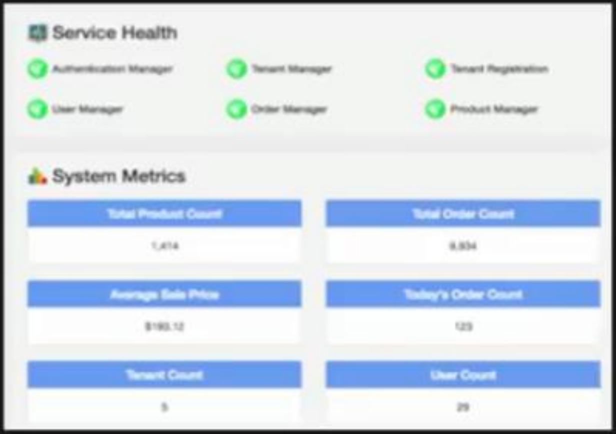
Extract role from token
Define view access policies
Show/hide views based on policies

Enabling client access ≠ enabling resource access



Metering consumption

Centralized system health



Service Health

🟢 Authentication Manager 🟢 Tenant Manager 🟢 Tenant Registration

🟢 User Manager 🟢 Order Manager 🟢 Product Manager

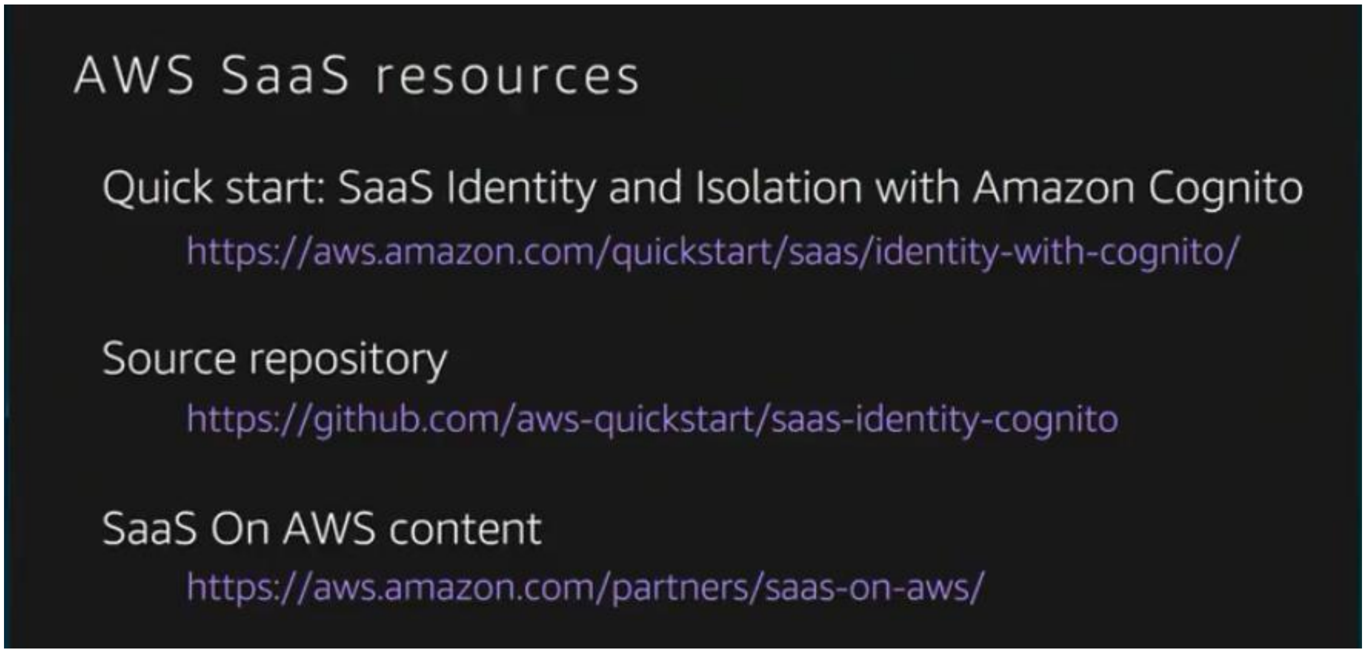
System Metrics

Total Product Count	Total Order Count
1,414	8,934
Average Sale Price	**Today's Order Count**
$193.12	123
Tenant Count	**User Count**
5	29

- Client polls services for system health
- Real-time view of microservice availability
- View of key activity metrics
- Services expose health endpoint
- Illustrates system vs. tenant flows

Takeaways

- Identity is foundational to SaaS application architecture
- IAM adds depth and power to your isolation model
- Limit developer awareness of security/tenant context
- Consider the impact of system *and* tenant roles
- Make metering and analytics an early priority
- Work the multitenant problem from end-to-end



AWS SaaS resources

Quick start: SaaS Identity and Isolation with Amazon Cognito
https://aws.amazon.com/quickstart/saas/identity-with-cognito/

Source repository
https://github.com/aws-quickstart/saas-identity-cognito

SaaS On AWS content
https://aws.amazon.com/partners/saas-on-aws/

AWS re:Invent

Thank you!