SaaS Identity and Isolation with Amazon Cognito on the AWS Cloud

Quick Start Reference Deployment

Judah Bernstein, Tod Golding, and Santiago Cardenas
Amazon Web Services

October 2017
Last updated: December 2017 (see revisions)
Contents

About This Guide ........................................................................................................................................... 4
About Quick Starts ........................................................................................................................................... 4
Quick Links .................................................................................................................................................... 4
Costs and Licenses ....................................................................................................................................... 5
Overview ...................................................................................................................................................... 5
Core SaaS Identity and Isolation Concepts ..................................................................................................... 5
Applying Roles .............................................................................................................................................. 8
System Users ................................................................................................................................................ 8
Tenant Users .................................................................................................................................................. 8
The Reference Application ............................................................................................................................... 9
Architecture ................................................................................................................................................... 9
Planning the Deployment ............................................................................................................................... 11
Prerequisites .................................................................................................................................................. 11
Deployment Options .................................................................................................................................... 11
Deployment Steps ......................................................................................................................................... 11
Step 1. Prepare Your AWS Account .............................................................................................................. 11
Step 2. Launch the Quick Start ..................................................................................................................... 12
Step 3. Test the Deployment ......................................................................................................................... 17
Walkthrough .................................................................................................................................................. 18
Onboarding Tenants ....................................................................................................................................... 18
Managing Tenant Users ................................................................................................................................. 20
Managing the Catalog .................................................................................................................................... 21
Managing Orders ........................................................................................................................................... 21
Performing System Administrator Functions ................................................................................................. 22
Implementation Details ................................................................................................................................ 23
Conceptual Architecture ............................................................................................................................... 23
Orchestrating Onboarding ............................................................................................................................. 24
Orchestrating Authentication ......................................................... 25
Binding Identity to Tenant and Policies ........................................ 26
Connecting The Dots ................................................................. 27
  Provisioning Policies .............................................................. 27
  Defining Custom Claims ......................................................... 28
Using Tokens .............................................................................. 29
Using the Custom Authorizer .................................................... 30
Putting it All Together ............................................................. 33
Considerations for Production Environments ............................. 34
Application Libraries ............................................................... 34
  Node.js Modules ................................................................. 34
  AngularJS Modules ............................................................ 35
  Python Modules ................................................................. 35
FAQ ......................................................................................... 35
Additional Resources .............................................................. 36
GitHub Repository ................................................................. 37
Document Revisions .............................................................. 37
About This Guide

This Quick Start reference deployment guide provides step-by-step instructions for deploying a solution for software-as-a-service (SaaS) identity and isolation with Amazon Cognito on the Amazon Web Services (AWS) Cloud.

SaaS architects can use the information in this guide to determine how best to build an end-to-end solution that integrates the core constructs of authentication and authorization with the policies of AWS Identity and Access Management (IAM) that define user roles and prevent cross-tenant access. The code included with this Quick Start also provides SaaS developers with a clear starting point for building their own SaaS identity solutions.

About Quick Starts

Quick Starts are automated reference deployments that use AWS CloudFormation templates to deploy a specific workload or solution on AWS, following AWS best practices.

Quick Links

The links in this section are for your convenience. Before you launch the Quick Start, please review the architecture, configuration, network security, and other considerations discussed in this guide.

- If you have an AWS account, and you’re already familiar with AWS services and SaaS identity and isolation concepts, you can launch the Quick Start to build the architecture shown in Figure 1 in a new or existing virtual private cloud (VPC). The deployment takes approximately 2 hours and 15 minutes. If you’re new to AWS or to SaaS, please review the overview and implementation details, and follow the step-by-step instructions provided later in this guide.

- If you want to take a look under the covers, you can view the AWS CloudFormation templates that automate the deployment.
Costs and Licenses
You are responsible for the cost of the AWS services used while running this Quick Start reference deployment. There is no additional cost for using the Quick Start.

The AWS CloudFormation template for this Quick Start includes configuration parameters that you can customize. For cost estimates, see the pricing pages for each AWS service you will be using. Prices are subject to change.

Overview
This Quick Start provides SaaS ISVs with a reference solution that illustrates architectural strategies for building the fundamentals of SaaS identity and isolation, using Amazon Cognito as the system’s underlying identity provider. This intent is to weave together all the moving parts that must be orchestrated by each SaaS provider to successfully address the full range of user and system requirements that are typically associated with building a robust SaaS identity model on AWS.

This Quick Start gives you a set of building blocks that can serve as a foundation for introducing core identity concepts into your own SaaS application. With the Quick Start code and the infrastructure that it deploys, you’ll get a first-hand look at SaaS identity best practices in a working application that connects all the fundamental concepts associated with identity and isolation in multi-tenant SaaS environments.

Core SaaS Identity and Isolation Concepts
In this section, we’ll explore some of the core concepts that shaped the solution deployed by this Quick Start. It’s important to understand how multi-tenancy influences your approach to building a robust and efficient identity and isolation model.

In traditional, single-tenant environments, identity tends to have a narrower scope where most of the focus is on managing a user’s credentials and personally identifiable information (PII). These single-tenant solutions typically depend on identity brokers and providers to implement their application’s authentication and authorization model. Meanwhile, in multi-tenant environments, users require additional identity data to accurately resolve the tenant context for a user.

The following diagram provides a conceptual mapping of these concepts. You’ll notice the traits that clearly distinguish user and tenant identities. Our goal in this solution is to create a unified notion of identity that joins these two concepts into what we have labeled a SaaS
identity. This joining of these identities enables the components of your system to have full access to tenant and user context.

In addition to thinking about how identity context is conveyed throughout your SaaS system, your solution must also consider how that identity will be used to constrain access to resources. Multi-tenant systems often leverage shared resources as part of their architecture, and, to eliminate any cross-tenant access to these resources, you’ll need to associate policies and roles with each user to scope their view of the system.

The following diagram highlights these concepts. Two tenants have access to a series of AWS resources. With security policies, each tenant should be allowed to access only their own resources.

![Figure 1: Creating a SaaS identity](image1)

![Figure 2: Preventing cross-tenant access](image2)
The key point here is that identity alone does not address the full security requirements of SaaS environments. While being authenticated certainly represents one barrier to entry, you’ll still want to rely on an added layer of policies that ensure that a given tenant never has the opportunity to access another tenant’s resources.

Any discussion of SaaS identity must also touch on roles. In SaaS environments, you have an extra set of dimensions you must consider when thinking about the roles that are associated with a user. For example, users may be system users or tenant users. A system user is an administrator of a SaaS provider and has access to all tenant data, whereas a tenant user is constrained to managing configuration and data that is part of their environment.

Figure 3 provides a conceptual view of these system and tenant roles. You can see that there are individual tenant administrators who manage their tenant users (each of whom could have their own distinct roles). There are also system roles, which are used to provision additional types of system users (operations, support, and so on). Users in the system role, for example, will have access to all the tenants who are part of the system. While the exact nature of these roles may change from system to system, your identity provisioning and isolation approach must accommodate the varying needs of these different user roles.

As we dig into the details of the Quick Start environment, you’ll find that the lifecycle and access profiles of these tenants introduce some special considerations. We’ll discuss how these different roles are provisioned and how these roles influence the underlying implementation of your identity and isolation model.
Applying Roles
Most applications include some notion of user roles. These roles define the context and scope of access for each user, and control access to a system’s features and resources. SaaS adds some complexity to this model, requiring roles that must accommodate the varying needs of both system and tenant users.

To help clarify the nature of these roles, the environment provisioned by the Quick Start supports a small collection of system and tenant users. These users are included to highlight the approaches that can be used to control a user’s access to application features and resources, and are discussed in the following sections.

System Users
A system user is a user who requires access to all tenants. These users represent the ISVs who provide varying types of management, monitoring, and support access to their SaaS tenants. In this category, you will typically find system administrators, operations personnel, and customer support staff with varying levels of access. For this Quick Start, we’ve included two specific roles to highlight the fundamentals of controlling system-level access.

- **System administrator** – This role has full access to all tenant environments, including the ability to introduce new system users into the system. This role also has access to system health data that others cannot see.

- **System user** – This role represents a support level user with more constrained access to the system’s data. This user has no ability to manage users or view system health, and can only view application data to help troubleshoot customer issues.

Tenant Users
A tenant user is a user who is scoped to a specific tenant. These users represent the individual tenant roles that are used to manage tenant configuration, and assume the various domain-specific roles of a given application. The initial tenant user, often referred to as the owner, is created during the registration process. All subsequent users are then created by this account. The Quick Start sets up the following two roles to highlight the variations in scoping tenant access:

- **Tenant administrator** – This role has full access to the tenant environment, including the ability to introduce new tenant users into the system. This role also has access to application functionality.

- **Tenant user** – This role represents an application-level user with more constrained access to tenant data and functionality. This user has no ability to manage users, and has limited access to application functionality.
The Reference Application

To effectively illustrate the concepts outlined in the previous section, the Quick Start deploys a reference application that provides an end-to-end SaaS identity and isolation solution.

The reference application represents a lightweight version of a SaaS order management system. It provides basic functionality for managing a catalog of products and support for creating orders for those products. This functionality helps highlight the different aspects of identity and isolation spanning the roles outlined in the previous section. After you deploy the Quick Start, you can follow the steps in the walkthrough to explore the flows that are supported by the application.

Architecture

Deploying this Quick Start for a new virtual private cloud (VPC) with default parameters builds the following SaaS identity and isolation environment in the AWS Cloud.

Figure 4: Deployed application architecture
This Quick Start’s architecture includes a number of AWS services and constructs, to create a highly scalable, highly available solution that conforms to best practices for deploying a container-based application in a VPC that spans two Availability Zones.

Let’s start by looking at the reference application, which is deployed using Amazon Simple Storage Service (Amazon S3). All of the assets of this AngularJS application are deployed to, and served from, an S3 bucket. The deployed web application interacts with the application’s back-end services through RESTful calls that are routed through Amazon API Gateway.

API Gateway provides a natural way to expose your services in SaaS environments, allowing you to better meter and throttle access to your environment. It also supports a custom authorizer that can validate the system’s tokens on each attempt to access services. This authorizer is implemented as an AWS Lambda function that allows you to create custom authorization logic for requests as they flow though the gateway.

Within the VPC, the architecture employs NAT gateways deployed in separate Availability Zones. These gateways, which are hosted in the VPC’s public subnets, provide high availability routing of traffic that flows from your private subnets to other AWS services or to the Internet.

The core of the application’s services are hosted in the VPC’s private subnets. An Amazon EC2 Container Service (Amazon ECS) cluster hosts the containers that run the system’s microservices. Seven separate Node.js microservices are deployed in this cluster. This cluster also employs Auto Scaling for basic high availability (HA). You can further tune this cluster to dynamically respond to changes in tenant load, scaling up and down based on demand.

On the right side of the architecture diagram, you’ll notice the various AWS services that the application uses. For example:

- DynamoDB tables are provisioned for some of the services that require storage.
- IAM manages and applies the application isolation polices and roles.
- Amazon Cognito serves as the identity provider.
- Amazon Simple Notification Service (Amazon SNS) publishes validation emails during the user registration process.
The architecture also supports continuous deployment: It uses a combination of AWS CodePipeline, AWS CodeBuild, S3 buckets, and the Amazon EC2 Container Registry (Amazon ECR) to manage the build and deployment of new application features.

Planning the Deployment

Prerequisites

Before you deploy this Quick Start, we recommend that you become familiar with the following AWS services. (If you are new to AWS, see Getting Started with AWS.)

- Amazon VPC
- Amazon ECS
- API Gateway
- DynamoDB
- IAM
- Amazon Cognito

Deployment Options

This Quick Start provides two deployment options:

- **Deploy the Quick Start into a new VPC** (end-to-end deployment). This option builds a new AWS environment consisting of the VPC, subnets, NAT gateways, security groups, and other infrastructure components, and then deploys the Quick Start into this new VPC.

- **Deploy the Quick Start into an existing VPC**. This option provisions the Quick Start in your existing AWS infrastructure.

The Quick Start also lets you configure additional settings that are discussed later in this guide.

Deployment Steps

Step 1. Prepare Your AWS Account

1. If you don't already have an AWS account, create one at [https://aws.amazon.com](https://aws.amazon.com) by following the on-screen instructions.

2. Use the region selector in the navigation bar to choose the AWS Region where you want to deploy this Quick Start on AWS.
Important This Quick Start uses Amazon Cognito, AWS CodeBuild, Amazon ECS, and Amazon ECR, which may not be available in all AWS Regions. Before you launch this Quick Start, please check the region table for availability.

3. Create a key pair in your preferred region.

Step 2. Launch the Quick Start

Note You are responsible for the cost of the AWS services used while running this Quick Start reference deployment. There is no additional cost for using this Quick Start. For full details, see the pricing pages for each AWS service you will be using in this Quick Start. Prices are subject to change.

1. Choose one of the following options to launch the AWS CloudFormation template into your AWS account. For help choosing an option, see deployment options earlier in this guide.

   Option 1
   Deploy Quick Start into a new VPC on AWS
   ![Launch button]

   Option 2
   Deploy Quick Start into an existing VPC on AWS
   ![Launch button]

Important If you’re deploying the Quick Start into an existing VPC, make sure that your VPC has two public and two private subnets in different Availability Zones. These subnets require NAT gateways or NAT instances in their route tables, to allow the instances to download packages and software without exposing them to the Internet. You’ll also need the domain name option configured in the DHCP options, as explained in the Amazon VPC documentation. You’ll be prompted for your VPC settings when you launch the Quick Start.

Each deployment takes about 2 hours and 15 minutes to complete.

2. Check the region that’s displayed in the upper-right corner of the navigation bar, and change it if necessary. This is where the network infrastructure for the Quick Start will be built. The template is launched in the US West (Oregon) Region by default.
Important  This Quick Start uses Amazon Cognito, AWS CodeBuild, Amazon ECS, and Amazon ECR, which may not be available in all AWS Regions. Before you launch this Quick Start, please check the region table for availability.

3. On the Select Template page, keep the default setting for the template URL, and then choose Next.

4. On the Specify Details page, change the stack name if needed. Review the parameters for the template. Provide values for the parameters that require input. For all other parameters, review the default settings and customize them as necessary. When you finish reviewing and customizing the parameters, choose Next.

In the following tables, parameters are listed by category and described separately for the two deployment options:

- Parameters for deploying the Quick Start into a new VPC
- Parameters for deploying the Quick Start into an existing VPC

Option 1: Parameters for deploying the Quick Start into a new VPC

View template

Network Configuration:

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability Zones (AvailabilityZones)</td>
<td>Requires input</td>
<td>The list of Availability Zones to use for the subnets in the VPC. The Quick Start uses two Availability Zones from your list and preserves the logical order you specify.</td>
</tr>
<tr>
<td>VPC CIDR (VPCCIDR)</td>
<td>10.0.0.0/16</td>
<td>CIDR block for the VPC.</td>
</tr>
<tr>
<td>Private Subnet 1 CIDR (PrivateSubnet1CIDR)</td>
<td>10.0.0.0/19</td>
<td>CIDR block for the private subnet located in Availability Zone 1.</td>
</tr>
<tr>
<td>Private Subnet 2 CIDR (PrivateSubnet2CIDR)</td>
<td>10.0.32.0/19</td>
<td>CIDR block for the private subnet located in Availability Zone 2.</td>
</tr>
<tr>
<td>Public Subnet 1 CIDR (PublicSubnet1CIDR)</td>
<td>10.0.128.0/20</td>
<td>CIDR block for the public (DMZ) subnet located in Availability Zone 1.</td>
</tr>
<tr>
<td>Public Subnet 2 CIDR (PublicSubnet2CIDR)</td>
<td>10.0.144.0/20</td>
<td>CIDR block for the public (DMZ) subnet located in Availability Zone 2.</td>
</tr>
</tbody>
</table>
### AWS Quick Start Configuration:

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quick Start S3 Bucket Name</strong> (QSS3BucketName)</td>
<td>aws-quickstart</td>
<td>S3 bucket where the Quick Start templates and scripts are installed. Use this parameter to specify the S3 bucket name you've created for your copy of Quick Start assets, if you decide to customize or extend the Quick Start for your own use. The bucket name can include numbers, lowercase letters, uppercase letters, and hyphens, but should not start or end with a hyphen.</td>
</tr>
<tr>
<td><strong>Quick Start S3 Key Prefix</strong> (QSS3KeyPrefix)</td>
<td>saas-identity-cognito/</td>
<td>The S3 key name prefix used to simulate a folder for your copy of Quick Start assets, if you decide to customize or extend the Quick Start for your own use. This prefix can include numbers, lowercase letters, uppercase letters, hyphens, and forward slashes.</td>
</tr>
</tbody>
</table>

### SaaS Identity Quick Start Configuration:

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Pair Name</strong> (KeyPairName)</td>
<td>Requires input</td>
<td>Public/private key pair, which allows you to connect securely to your instance after it launches. When you created an AWS account, this is the key pair you created in your preferred region. (Leaving this field blank will result in an AWS CloudFormation failure during the Amazon ECS container stack creation.)</td>
</tr>
<tr>
<td><strong>Instance Type</strong> (InstanceType)</td>
<td>t2.small</td>
<td>EC2 instance type for the Amazon ECS host. The three options are t2.micro, t2.small, and t2.large.</td>
</tr>
</tbody>
</table>

### SaaS Identity Sys Admin Configuration:

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Email Address</strong> (Email)</td>
<td><a href="mailto:email@example.com">email@example.com</a></td>
<td>The email address of the initial systems admin user. After deployment, an email with a temporary password is sent to this address.</td>
</tr>
<tr>
<td><strong>Company Name</strong> (Company)</td>
<td>My Company Name</td>
<td>The company name of the user who is deploying the Quick Start.</td>
</tr>
<tr>
<td><strong>First Name</strong> (Firstname)</td>
<td>MyFirstName</td>
<td>The first name of the initial systems admin user.</td>
</tr>
<tr>
<td><strong>Last Name</strong> (Lastname)</td>
<td>MyLastName</td>
<td>The last name of the initial systems admin user.</td>
</tr>
</tbody>
</table>
**SaaS Identity DynamoDB Table Configuration:**

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User (UserTable)</td>
<td>User</td>
<td>The name of the DynamoDB table that contains the users of the multi-tenant system (maximum of 36 characters).</td>
</tr>
<tr>
<td>Tenant (TenantTable)</td>
<td>Tenant</td>
<td>The name of the DynamoDB table that contains the tenants of the multi-tenant system (maximum of 36 characters).</td>
</tr>
<tr>
<td>Product (ProductTable)</td>
<td>Product</td>
<td>The name of the DynamoDB table that contains the products of the multi-tenant system (maximum of 36 characters).</td>
</tr>
<tr>
<td>Order (OrderTable)</td>
<td>Order</td>
<td>The name of the DynamoDB table that contains the orders of the multi-tenant system (maximum of 36 characters).</td>
</tr>
</tbody>
</table>

- **Option 2: Parameters for deploying the Quick Start into an existing VPC**

  **View template**

  **Network Configuration:**

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC ID (VPC)</td>
<td>Requires input</td>
<td>ID of your existing VPC (e.g., vpc-0343606e).</td>
</tr>
<tr>
<td>Private Subnet 1 ID (PrivateSubnet1)</td>
<td>Requires input</td>
<td>ID of the private subnet in Availability Zone 1 in your existing VPC (e.g., subnet-a0246dcd).</td>
</tr>
<tr>
<td>Private Subnet 2 ID (PrivateSubnet2)</td>
<td>Requires input</td>
<td>ID of the private subnet in Availability Zone 2 in your existing VPC (e.g., subnet-b58c3d67).</td>
</tr>
<tr>
<td>Public Subnet 1 ID (PublicSubnet1)</td>
<td>Requires input</td>
<td>ID of the public subnet in Availability Zone 1 in your existing VPC.</td>
</tr>
<tr>
<td>Public Subnet 2 ID (PublicSubnet2)</td>
<td>Requires input</td>
<td>ID of the public subnet in Availability Zone 2 in your existing VPC.</td>
</tr>
</tbody>
</table>

  **AWS Quick Start Configuration:**

<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Start S3 Bucket Name (QSS3BucketName)</td>
<td>aws-quickstart</td>
<td>S3 bucket where the Quick Start templates and scripts are installed. Use this parameter to specify the S3 bucket name you’ve created for your copy of Quick Start assets, if you decide to customize or extend the Quick Start for your own use. The bucket name can include numbers, lowercase letters, uppercase letters, and hyphens, but should not start or end with a hyphen.</td>
</tr>
</tbody>
</table>
Parameter label (name) | Default | Description
---|---|---
**Quick Start S3 Key Prefix** (QSS3KeyPrefix) | saas-identity-cognito/ | The S3 key name prefix used to simulate a folder for your copy of Quick Start assets, if you decide to customize or extend the Quick Start for your own use. This prefix can include numbers, lowercase letters, uppercase letters, hyphens, and forward slashes.

**SaaS Identity Quick Start Configuration:**

Parameter label (name) | Default | Description
---|---|---
**Key Pair Name** (KeyPairName) | Requires input | Public/private key pair, which allows you to connect securely to your instance after it launches. When you created an AWS account, this is the key pair you created in your preferred region. (Leaving this field blank will result in an AWS CloudFormation failure during the Amazon ECS container stack creation.)

**Instance Type** (InstanceType) | t2.small | EC2 instance type for the Amazon ECS host. The three options are t2.micro, t2.small, and t2.large.

**SaaS Identity Sys Admin Configuration:**

Parameter label (name) | Default | Description
---|---|---
**Email Address** (Email) | email@example.com | The email address of the initial systems admin user. After deployment, an email with a temporary password is sent to this address.

**Company Name** (Company) | My Company Name | The company name of the user who is deploying the Quick Start.

**First Name** (Firstname) | MyFirstName | The first name of the initial systems admin user.

**Last Name** (Lastname) | MyLastName | The last name of the initial systems admin user.

**SaaS Identity DynamoDB Table Configuration:**

Parameter label (name) | Default | Description
---|---|---
**User** (UserTable) | User | The name of the DynamoDB table that contains the users of the multi-tenant system (maximum of 36 characters).

**Tenant** (TenantTable) | Tenant | The name of the DynamoDB table that contains the tenants of the multi-tenant system (maximum of 36 characters).
<table>
<thead>
<tr>
<th>Parameter label (name)</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product (ProductTable)</td>
<td>Product</td>
<td>The name of the DynamoDB table that contains the products of the multi-tenant system (maximum of 36 characters).</td>
</tr>
<tr>
<td>Order (OrderTable)</td>
<td>Order</td>
<td>The name of the DynamoDB table that contains the orders of the multi-tenant system (maximum of 36 characters).</td>
</tr>
</tbody>
</table>

5. On the **Options** page, you can **specify tags** (key-value pairs) for resources in your stack and **set advanced options**. When you’re done, choose **Next**.

6. On the **Review** page, review and confirm the template settings. Under **Capabilities**, select the check box to acknowledge that the template will create IAM resources.

7. Choose **Create** to deploy the stack.

8. Monitor the status of the stack. When the status is **CREATE_COMPLETE**, the Quick Start cluster is ready.

9. Use the URLs displayed in the **Outputs** tab for the stack to view the resources that were created.

**Step 3. Test the Deployment**
Once the deployment completes successfully, you can begin to use the provisioned environment.

1. Find the URL of the generated website in the **Outputs** tab for your CloudFormation stack, as shown in Figure 5, and open it in your browser.

![Figure 5: Locating the website URL](image-url)
2. To log in to the reference application, use the email address you specified during deployment (**Email Address** parameter), which represents a system administrator) and the temporary password you receive in email. The system is now ready for a new tenant to begin the onboarding process.

**Walkthrough**

The walkthrough in this section provides a detailed view of how the reference application manages and applies each of the SaaS user roles described in the overview.

Let’s explore the different flows that are supported by the application.

**Onboarding Tenants**

Many SaaS applications provide their users with a frictionless onboarding process that allows them to create new accounts. Although this process is usually relatively straightforward for new tenants, the underlying implementation and automation can be quite involved.

With tenant registration, the application needs to successfully provision a new tenant’s identity, configure their account, and go through a validation process. The successful completion of this process results in the provisioning and configuration of the infrastructure for creating the tenant’s identity footprint, and the supporting policies that will manage the isolation policies associated with the new tenant.

The workflow for the registration process is shown in Figure 6.
When the application loads, you will be directed to a login form. From that form, choose **Register** to display the registration form. This form collects all the data needed to identify and register the tenant and the user who will be designated as the owner or primary user for that tenant. This screen also collects information about the plan or tier that the user is selecting. The tier is only shown here as a placeholder. At this point, the application doesn’t actually apply any logic to alter the user experience for each tier.

When you’ve completed the registration process, you’ll get a **Success** message, indicating that you’ve registered successfully. Next, you’ll get a message that welcomes you to the system and provides you with a temporary password for your initial login.

Now, return to the application’s login form, enter your email address and the temporary password that was provided, and choose **Login**. If you log in successfully, you’ll be prompted to enter a new password for your account. After you supply this information, your registration process will be complete and you’ll be granted access to the system.

You’ll now see the application’s home page, which provides navigation options and includes a dashboard that is populated with key system metrics. A sample image of this dashboard is shown in Figure 7.
The metrics shown in Figure 7 are simply placeholders to represent metrics that could be of value in an order management system. Ultimately, these would be replaced by live data.

**Managing Tenant Users**

As a newly registered tenant, you will enter the system in the role of tenant administrator. This role grants you full access to tenant functionality. It also gives you access to administration functions that would not be available to tenant users. Note that the tenant home page shown in Figure 7 includes a **Users** option in the upper right.

Choose this option to display a list of users for the logged in tenant. This list provides a number of options that would be typically included as part of a user management system. In this case, we’ve included options to edit, delete, and enable or disable individual users. Figure 8 provides an example of a tenant user list.
Managing the Catalog

Before you can accept orders, the system must be able to create and manage the catalog of available products. To support this, the system includes all the functionality you’ll need to add, edit, and delete products from a catalog. Figure 9 provides an example of the list of products that have been entered into the catalog. You can use the buttons to the right of each item to edit or delete catalog entries. You can also use the hyperlink in the SKU column to access a detailed view of a catalog item. Initially, your catalog will be empty. You can add new products by choosing the Add Product button in the upper right.

<table>
<thead>
<tr>
<th>SKU</th>
<th>Title</th>
<th>Condition</th>
<th>In Stock</th>
<th>Unit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>NIKE-91431</td>
<td>Nike Cobra Men’s Fly Z Driver</td>
<td>Refurbished</td>
<td>5</td>
<td>$129.00</td>
</tr>
<tr>
<td>TM-482029</td>
<td>TaylorMade Men’s M1 460cc Driver</td>
<td>Brand New</td>
<td>12</td>
<td>$399.00</td>
</tr>
<tr>
<td>ADM-39819</td>
<td>Adams Blue Hybrid Iron Set Womens</td>
<td>Scratch and Dent</td>
<td>1</td>
<td>$145.00</td>
</tr>
</tbody>
</table>

Figure 9: Catalog management

The ability to manage the catalog will vary based on the tenant role of the logged in user. If you’re a tenant administrator, you’re granted full access to the catalog. If you’re a tenant user, you can only view catalog items.

Managing Orders

After you add products to your catalog, you can begin to create orders for those items by selecting Orders from the application’s navigation bar. To add an order, you can select from available products. The Orders page displays each new order with an order date, a quantity, and a price, as shown in Figure 10. You can filter this list by product, and edit or delete orders from this screen.

Figure 10: Order management
Performing System Administrator Functions
The onboarding, tenant user management, catalog and order management workflows that we’ve outlined so far have focused exclusively on the tenant administration and tenant user roles. For system users, the application intentionally excludes any public onboarding process. Instead, this Quick Start automatically provisions a system administrator user for the demo environment by using the email address and other information you specify during deployment. You can use this user role to manage and create additional system users in the different roles that are supported.

To highlight the role of the system administrator, we’ve augmented the application’s dashboard, adding new elements that reflect the overall health of the system. Figure 11 illustrates this dashboard in action. It includes both the tenant metrics that were discussed earlier and the health of the system’s microservices. Each microservice has a corresponding indicator that reflects the health of the service. In Figure 11, all the service indicators are green, meaning they are all healthy. If one of these services were to fail, its health indicator would change to red.

Figure 11: System Admin home page
Implementation Details

Conceptual Architecture

Now that we’ve discussed the application flows and roles, let’s look at the underlying design and implementation that support these capabilities. This Quick Start focuses on SaaS identity and isolation, but also aligns with cloud native application models where appropriate, so it includes a set of microservices that are running in an Amazon ECS container. Figure 12 shows a highly conceptualized version of the architecture. (For a detailed architecture diagram, see Figure 1 earlier in this guide.)

![Conceptual architecture diagram]

The architecture relies on a handful of AWS technologies. The reference application uses AngularJS with content served directly from an S3 bucket. Requests from the application are routed through the API Gateway and directed to each microservice that supports the application. The microservices, developed with Node.js, are executed in an Amazon ECS container cluster.

Some of these services rely on DynamoDB for storage. Others don’t have any storage that they directly own. Amazon Cognito is also used throughout the system to provide the fundamental user management infrastructure. Finally, you’ll notice that a custom authorizer is connected to the gateway, and provides validation of tokens that flow through this experience.
Orchestrating Onboarding

Let's look at how the tenant onboarding process orchestrates the moving parts of the system’s architecture to create a new tenant in the system. Figure 13 outlines the interactions between the various elements of the Quick Start architecture.

![Figure 13: Orchestrating Onboarding](image)

The onboarding workflow consists of these steps:

1. The new tenant completes and submits the registration form.
2. Registration confirms that the user doesn’t exist.
3. User management creates a new user in its local database.
4. User management creates a new user pool and custom claims in Amazon Cognito.
5. Tenant management creates a new tenant.
6. Tenant registration provisions policies for each tenant role.

When this process is finished, all the moving parts of a new tenant will have been successfully provisioned. The tenant user will now exist and be ready for authentication in Amazon Cognito. The tenant and all its required policies will have been provisioned.
Orchestrating Authentication

Once a user is created, you need to look at how that user is authenticated by the system. The authentication process connects users to tenants, and gets the context that limits and controls users’ access to resources.

Fortunately, most of this process can be orchestrated through a series of calls to the Amazon Cognito API using what is referred to as the Enhanced (Simplified) Flow. Figure 14 provides an illustration of this flow, as implemented by this Quick Start.

The authentication flow is orchestrated entirely by the system’s authentication service and consists of these steps:

1. The authentication service determines if the user exists, and, if so, determines their assigned user pool ID.
2. The system retrieves information about that pool from Amazon Cognito. At this stage, the service has the parameters it needs to authenticate the user against a specific user pool.
3. The user is now authenticated against their assigned user pool, and successful authentication returns an ID token.
4. The ID token is used to get a Cognito ID from Amazon Cognito.

5. Amazon Cognito receives a request for a temporary STS token from the STS service, exchanges the supplied Cognito ID for an STS token, and returns that token back to the authentication service.

6. The authentication service returns the ID token and the STS token (used as an access token) back to the client. The ID token supplies the client with access to the claims data about the user. Both the ID and the STS token are passed through in calls to downstream servicing. They supply both the tenant context for the user and the access ID and secret keys that will be used to access AWS resources.

**Binding Identity to Tenant and Policies**

The onboarding process creates the fundamental bindings that are essential to the SaaS identity and isolation solution deployed by this Quick Start. Figure 15 shows the three components of this relationship.

![Figure 15: Binding tenants and roles to identity](image)

With Amazon Cognito, we create a user identity and we introduce attributes into that identity that describe its relationship to tenants. So, in this case, the Quick Start provisions and populates a tenant identifier and a role with each user. This tenant identifier is then connected with the user via this identifier. This connects the user to tenant attributes like the billing tier and status (active/inactive). Finally, the tenant identifier connects the user and the tenant to a collection of IAM policies that enforce tenant isolation.

The ability to connect these three concepts into a single, unified model is a key aspect of this Quick Start solution. Identifying a user’s relationship with a tenant and mapping that to enforceable policies enable developers to address many of the identity and isolation goals outlined earlier in this guide.
Connecting The Dots

The following sections provide a more granular view of how the Quick Start implements the key elements of the system.

Provisioning Policies

This Quick Start allows the system to bind a tenant to a set of policies that determine the tenant’s scope of access to AWS resources. When a new tenant is onboarded (following the process outlined previously), the system provisions the policies for that tenant. This Quick Start uses DynamoDB to illustrate how these generated policies could be used to demonstrate isolation. The data for each tenant is stored in a series of DynamoDB tables. These tables use a pooled multi-tenant model where each tenant’s data lives with data from other tenants within each table. With this model, you need some way to enforce isolation and prevent cross-tenant access within these tables.

Now let’s consider how the system can create policies that achieve these isolation goals. With DynamoDB, the Quick Start relies on the IAM service’s ability to define policies that filter access to specific items in a DynamoDB table. Figure 16 shows a sample policy that constrains tenant access to the items in a table.

Figure 16: Scoping tenant access with policies
This example outlines a set of operations that can be performed on the system’s Order table. The key item to focus on is the `dynamodb:leadingkeys` value, which includes a tenant identifier. This value limits the view of data to only those items in the table that have a matching tenant ID as the prefix to a table’s partition key value.

To limit the bloat of policies and keep your environment manageable, the Quick Start introduces a unique set of policies and roles for each tenant. This creates a much smaller footprint, eliminating the need for separate policies for each tenant.

Note that the Quick Start focuses fairly exclusively on demonstrating isolation for DynamoDB tables. In a more complete system, you would need to extend these scoping policies to a much broader range of AWS resources, each of which will require its own model for achieving isolation.

**Defining Custom Claims**

This Quick Start uses claims to represent attributes that associate tenant information with each user. These claims are packaged and transported in an encoded JSON Web Token (JWT). These tokens contain data that is pulled from the attributes defined in the Amazon Cognito user pool for a tenant. Amazon Cognito supports a standard set of attributes that are supported by all implementations of the OpenID Connect (OIDC) protocol. However, these standard attributes don’t include some of the items you need to associate a user with a tenant. These additional data elements must be introduced as custom attributes, which are then conveyed as custom claims in the JWT returned by the authentication process.

The custom attributes are provisioned and configured when each tenant is onboarded to the system. The Quick Start solution first creates the tenant’s user pool, and then adds the required attributes to this pool. The registration process collects and generates the data for these attributes, and injects them into the user provisioning process. Then, when the user is authenticated, these attributes are encoded and returned as an ID token.

Although the custom claims are supported by the OpenID Connect standard, using them for your SaaS solution does represent a tradeoff. When you use custom claims, your solution will be bound to a single identity provider (in this case, Amazon Cognito). If, for any reason, you want to switch to another provider (for example, LinkedIn or Facebook), your new provider would not be aware of these custom claims. So, while custom claims provide a natural mechanism for conveying tenant attributes, you’ll need to factor this limitation into your strategy. In some cases, you may need to rely on another pattern to bind a user’s identity to a tenant.
Using Tokens
This Quick Start solution uses tokens to convey information about tenants and their access privileges. A successful authentication will yield two tokens: an ID token and an access token. These tokens are returned to the client in a JWT format and are encoded as they are passed to and from the client. Each consumer must decode the contents of each token to access the JSON object that contains the attributes associated with that token.

The following code provides an example of how the client application decodes and displays the ID tokens that it receives.

![Code Example]

Figure 17: Decoding tokens on the client

These tokens are also inserted into the header of each HTTP request that is sent to the application’s services. Each service that processes a request must consult the ID token to acquire the tenant context. Also, if the service needs to access AWS resources, it must extract AWS credentials from the access token.

The code in Figure 18 provides an example of how these tokens are processed by the application services. This particular code focuses on the Product service that provides basic REST operations for managing product information.

![Code Example]

Figure 18: Extracting the tenant ID from the ID token

This code processes every incoming request that is sent to the service. It centralizes the extraction of the tenant ID from the bearer token. Using this approach allows you to have
one block of code to fetch the tenant ID that is referenced by each entry point implemented by a service.

You’ll notice that this function relies on the `tokenManager` to retrieve the tenant ID. This function is packaged in a centralized helper module to simplify and unify the application’s processing of token information.

We also need credentials to access the DynamoDB table for products. Figure 19 shows an example of how the Quick Start solution acquires these credentials.

```javascript
app.get('/product/:id', function(req, res) {
  winston.debug('Fetching product: ' + req.params.id);

  tokenManager.getCredentialsFromToken(req, function(creds) {
    // init params structure with request params
    var params = {
      tenantId: parseInt(req.params.tenantId,
      productId: req.params.productId
    }

    << remaining Get code >>
  });
});
```

**Figure 19: Getting tenant-scoped credentials**

This function implements the GET method for the Product REST service. The function calls `getCredentialsFromToken()`, passing in the HTTP request that includes the bearer token. This function extracts the tokens from the header, decodes them, and returns a credentials object that holds the attributes you’ll need for your DynamoDB API calls.

The tokens that flow through this process provide the tenant context required for scoping access along with the credentials that are scoped by IAM policies. With these elements in place, you have a reusable mechanism that can flow context through all the system’s services with minimal overhead. It also eliminates any latency that would be associated with resolving this context via other services.

**Using the Custom Authorizer**

After your application has acquired tokens from the authentication process, you’ll likely still want to validate those tokens as you make requests to your backend services. Fortunately, API Gateway provides a natural mechanism for injecting custom processing logic, referred to as a custom authorizer, to handle this validation.
Figure 20 provides an overview of how this custom validation is implemented by the Quick Start reference application. Notice that each request from the client is routed through API Gateway, which then maps these calls to their corresponding microservice functions. To create this mapping, you must connect API Gateway’s HTTP entry points to specific methods of the application’s backend services. During this configuration you can also introduce and configure your custom authorizer by configuring the settings for your HTTP method request and selecting the **Custom Authorizer** option for authorization. Figure 21 shows how you would access these options within the API Gateway console.
After you select the custom authorizer option, you can edit the settings for that authorizer, as illustrated in Figure 22. This includes the configuration of the Lambda function that will be used to validate tokens that are associated with each request.
The Lambda function that implements this Quick Start’s authorizer downloads the PEM certificates required for the Amazon Cognito user pool, which were initially used to sign the generated JWT. After the authorizer downloads the PEM certificates, it decodes the claims; verifies the issuer, the type of token (as distinguished by a claim within the JWT), and the signature of the provided identity token; and issues approvals for the specified resource methods.

This solution doesn’t support caching of PEM certificates for tenant user pools. If you’re using the solution in a production environment, you’ll likely want to add this optimization. This would be a natural place to leverage Amazon ElastiCache to improve the performance of your custom authorizer. Additionally, the custom authorizer is currently invoked on a per-method basis, with a policy set to * for all methods of the API. We recommend that you minimize the footprint of your policy by assigning specific IAM policies to the endpoint.

**Putting it All Together**

Let’s take a quick look at how these details fit into the overall identity and isolation model. Figure 23 highlights the connections between the core concepts that we discussed.

![Figure 23: Mapping Flows to Activities](image)

The three main workflows of the sample application are shown on the left, and the activities that are associated with each flow are shown on the right. The tenant onboarding and registration process provisions the tenant and its corresponding policies. It also creates the user and the custom attributes it needs to bind to the tenant. Authentication associates a user with the appropriate ID and access tokens that convey a tenant’s access and authorization privileges. The tokens are returned to the client in encoded JWT format.
In the third dataflow, these tokens are injected into the service calls to the application services. The ID and access tokens are passed in HTTP headers to each service, where they are verified by API Gateway, decoded by the microservice, and used to resolve the current tenant context and acquire credentials to access AWS resources. These credentials are then passed directly to each AWS service client, such as Amazon DynamoDB, each time the code is invoked.

**Considerations for Production Environments**

This Quick Start pulls together a complete end-to-end identity and isolation solution. However, if you’re planning to use this code in a production environment, we recommend that you consider adding the following functionality:

- Client certificates – For full security, we recommend that you use client certificates for authentication.

- Auto Scaling – The current solution is employed in a Multi-AZ model for high availability. However, there are no policies attached to the Auto Scaling model that measure and respond to load changes on the services. Consider tuning this area based on the specific scaling dimensions of the services in your domain.

- JWT encoding – The tokens that are passed through the system aren’t currently encoded. To better secure the content of your tokens, we recommend that you use an encryption library.

- API keys – Some services might need to expose methods as part of a developer API. You should secure these entry points with API keys, which are supported by API Gateway.

- Microservice communications – In the current implementation, all interactions from the client to a microservice pass through API Gateway. However, some calls between microservices are handled as direct HTTP calls. To better scale the system, consider routing these calls back through API Gateway or handle them asynchronously, for example, by using Amazon Simple Queue Service (Amazon SQS).

**Application Libraries**

The Quick Start application relies on the following open-source frameworks and tools.

**Node.js Modules**

- Async
- `amazon-cognito-identity-js`
- Config
- Winston
- UUID
• body-parser
• aws-sdk
• amazon-cognito-js
• Express
• Randomstring
• Request
• jsonwebtoken
• jwk-to-pem
• jwt-decode

AngularJS Modules
• Angular
• Bootstrap
• angular-route
• angular-jwt
• angular-mocks
• Grunt
• Karma
• phantomjs-prebuilt
• time-grunt
• autoprefixer-core

Python Modules
• JSON
• Requests
• cfn-response
• Boto 3

FAQ

Q. I encountered a CREATE_FAILED error when I launched the Quick Start. What should I do?

A. If AWS CloudFormation fails to create the stack, we recommend that you relaunch the template with **Rollback on failure** set to **No**. (This setting is under **Advanced** in the AWS CloudFormation console, **Options** page.) With this setting, the stack’s state will be retained and the instance will be left running, so you can troubleshoot the issue. (You’ll want to look at the log files in `%ProgramFiles%\Amazon\EC2ConfigService` and `C:\cfn\log`.)
### Important

When you set **Rollback on failure** to No, you’ll continue to incur AWS charges for this stack. Please make sure to delete the stack when you’ve finished troubleshooting.

For additional information, see [Troubleshooting AWS CloudFormation](https://aws.amazon.com/documentation/cloudformation/) on the AWS website.

**Q.** I encountered a size limitation error when I deployed the AWS Cloudformation templates.

**A.** We recommend that you launch the Quick Start templates from the location we’ve provided or from another S3 bucket. If you deploy the templates from a local copy on your computer or from a non-S3 location, you might encounter template size limitations when you create the stack. For more information about AWS CloudFormation limits, see the [AWS documentation](https://aws.amazon.com/documentation/cloudformation/).

### Additional Resources

**AWS services**

- Amazon Cognito  
  [https://aws.amazon.com/documentation/cognito/](https://aws.amazon.com/documentation/cognito/)
- Amazon DynamoDB  
  [https://aws.amazon.com/documentation/dynamodb/](https://aws.amazon.com/documentation/dynamodb/)
- Amazon EC2  
- Amazon VPC  
  [https://aws.amazon.com/documentation/vpc/](https://aws.amazon.com/documentation/vpc/)
- API Gateway  
  [https://aws.amazon.com/documentation/apigateway/](https://aws.amazon.com/documentation/apigateway/)
- AWS CloudFormation  
  [https://aws.amazon.com/documentation/cloudformation/](https://aws.amazon.com/documentation/cloudformation/)
- IAM  
  [https://aws.amazon.com/documentation/iam/](https://aws.amazon.com/documentation/iam/)
Quick Start reference deployments

- AWS Quick Start home page
  https://aws.amazon.com/quickstart/

GitHub Repository

You can visit our GitHub repository to download the templates and scripts for this Quick Start, to publish your comments, and to share your customizations with others.

Document Revisions

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>In sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2017</td>
<td>Added Instance Type parameter</td>
<td>Parameter tables in step 2</td>
</tr>
<tr>
<td>October 2017</td>
<td>Initial publication</td>
<td>—</td>
</tr>
</tbody>
</table>

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS’s current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS’s products or services, each of which is provided “as is” without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

The software included with this paper is licensed under the Apache License, Version 2.0 (the “License”), unless indicated otherwise. You may not use this file except in compliance with the License. A copy of the License is located at http://aws.amazon.com/apache2.0/ or in the "license" file accompanying this file. This code 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.