SaaS Solutions on AWS

Tenant Isolation Architectures

January 2016

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.

Contents

Abstract

Increasingly, the mode of delivery for enterprise solutions is turning toward the software as a service (SaaS) model, but architecting a SaaS solution can be challenging. There are multiple aspects that need to be taken care of, and a variety of options for deploying SaaS solutions on AWS. This paper covers the different SaaS deployment models and the combination of AWS services and AWS Partner Network (APN) partner solutions that can be used to achieve a scalable, available, secure, performant, and cost-effective SaaS offering.

AWS now offers a structured AWS SaaS Partner Program to help you build, launch, and grow SaaS solutions on AWS. As your business evolves, AWS will be there to provide the business and technical enablement support you need. Please review the SaaS Partner Program website for more details.

Introduction

There are a variety of solutions that can be deployed in a SaaS model, and these share a number of similarities and common patterns. In this paper, we will discuss:

- Common solution components – These are aspects that we recommend handling separately from the core, solution-related functional components, such as billing, monitoring, and analytics. We will discuss these components in detail.

- SaaS solution - tenant isolation architecture patterns – A solution can be deployed in multiple ways on AWS. We will discuss typical models that help with the requirements around a multi-tenant SaaS deployment, along with considerations for each of those cases.

This white paper focuses on the technology and architecture aspects of SaaS deployments, and does not attempt to address business and process-related aspects, such as software vendor licensing, SLAs, pricing models, and DevOps practice considerations.

Common Solution Components

In addition to building the core functional components of your SaaS solution, we highly recommend that you build additional supporting components that will help in future-proofing your solution and making it easier to manage. Building additional supporting components will also enable you to easily grow and add more tenants over time. The following sections discuss some of the recommended supporting components for SaaS solution setups.

Security and Networking (Tenant Isolation Modeling)

The first step in any multi-tenant system design is to define a strategy to keep the tenants secure and isolated from one another. This may include security considerations such as defining segregation at the network/storage layer, encrypting data at rest or in transit, managing keys and certificates safely, and even managing application-level security constructs. There are a number of AWS services you can use to help address security considerations at each level, including AWS CloudHSM, AWS CloudTrail, Amazon VPC, AWS WAF, Amazon Inspector, Amazon CloudWatch and Amazon CloudWatch Logs. By using native AWS services such as these, you can define a model that matches the solution's security and networking requirements. In addition to AWS native services, many customers also make use of APN Partner offerings in the infrastructure security space to augment their security posture, and add capabilities like intrusion detection systems (IDS)/intrusion prevention systems (IPS).

Identity Management, User Authentication, and Authorization

It's important to decide on the strategy for authenticating and authorizing users to manage both the AWS services and the SaaS application itself. For AWS services, you can use AWS Identity and Access Management (IAM) users, IAM roles, Amazon Elastic Compute Cloud (Amazon EC2) roles, social identities, directory/LDAP users, and even federated identities using SAML-based integrations. Likewise, for your application, you have multiple ways to authenticate users. We recommend building a layer that supports your application authentication requirements. You might consider Amazon Cognito-based authentication for mobile users, and you can also look to APN Partner offerings in the identity and access control space for managing authentication across different identity providers.

Monitoring, Logging, and Application Performance Management

You should have monitoring enabled at multiple layers, not only to help diagnose issues, but also to enable proactive measures to avoid issues down the road. You can benefit from utilizing the data from Amazon CloudWatch, which enables detailed monitoring for critical infrastructure, and lets you configure alarms to notify you of any issues. You could also make use of AWS Config that provides you with an AWS resource inventory, configuration history, and configuration change notifications to enable security and governance. For application-level monitoring, you could use the Amazon CloudWatch Logs functionality to stream the logs in real time to the service; in addition, you can search for patterns, and you can also track the number of errors that occur in your application logs and configure Amazon CloudWatch to send you a notification whenever the rate of errors exceeds a threshold you specify. Many companies also use APN Partner offerings in the logging and monitoring space to monitor application performance aspects.

Analytics

Most SaaS solutions have a wealth of raw data, including application logs, user access logs, and billing-related data, which generally can provide a lot of insight if properly analyzed. In addition to batch-oriented analysis, you can do real-time analytics to see what kind of actions are being invoked by various tenants on the platform, or look at real-time infrastructure-related metrics to detect any unexpected behavior and to preempt any future problems. You can use AWS services such as Amazon Elastic MapReduce (Amazon EMR), Amazon Redshift, Amazon Kinesis, Amazon Machine Learning, Amazon QuickSight, Amazon Simple Storage Service (Amazon S3), and Amazon EC2 Spot Instances to build these types of capabilities. Analytics is normally an ancillary function of a platform in the early stages, but as soon as multiple tenants are on-boarded to a SaaS platform, analytics quickly becomes a core function for detecting and understanding usage patterns, providing recommendations, and driving decisions. We recommend that you plan for this layer early in the solution development cycle. Figure 1 shows some of the AWS big data services and their capabilities, ranging from data ingestion to storage to data analytics/processing.

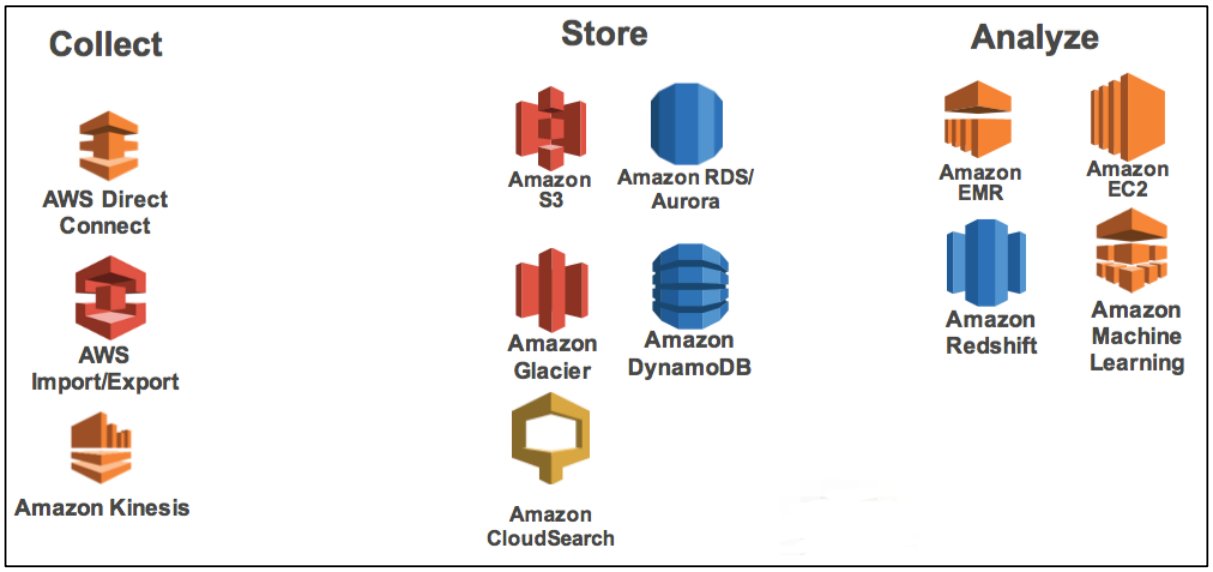

Figure 1: AWS Big Data and Analytics Services

Configuration Management and Provisioning

AWS provides a number of possibilities for automating solution deployments. You have the ability to bake some deployment tasks within the Amazon Machine Images (AMIs) themselves, and you can automate more configurable or frequent changes using various other means:

- One-time tasks like OS hardening or setting up specific versions of run-time environments that do not change without an application re-certification process (like a Java upgrade), or even time-consuming installations (like middleware/database setup) can be baked into the AMI itself.
- To handle more frequently changing aspects of deployment, like code updates from a code repository, boot-time tasks (like joining a domain/cluster), and certain environment-specific configurations (like different parameters for dev/test/production), you can use custom scripts in the EC2 instance's user data section or AWS services such as AWS CodeCommit, AWS CodePipeline, and AWS CodeDeploy.
- For complete stack spin-up, a higher level of automation can be achieved by using AWS CloudFormation, which gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, and enables them to provision and update those resources in an orderly and predictable

fashion. Depending on your requirements, AWS Elastic Beanstalk and AWS OpsWorks can also help with quick deployments and automation.

With the right mix of segregation across different types of tasks, you can achieve the correct balance between faster boot time (often needed for auto scaled layers) and a configurable, automated setup (needed for flexible deployments).

Storage, Backup, and Restore Capabilities

Most AWS services have mechanisms in place to perform backup so that you can revert to a last known stable state if any newer changes need to be backed out. Features, including Amazon EC2 AMI creation or snapshotting (Amazon EBS, Amazon RDS, and Amazon Redshift snapshots) can potentially support a majority of backup requirements. However, for advanced needs, such as the need to quiesce a file system and then take a consistent snapshot of an active database, you can use third-party backup tools, many of which are available on AWS Marketplace.

AWS Tagging Strategy

To help you manage instances, images, and other Amazon EC2 resources, you can assign your own metadata to each resource in the form of tags. We recommend that you adopt a tagging strategy before you begin to roll out your SaaS solution. Each tag consists of a key and an optional value, both of which you define. You can also have multiple tags on a single resource. There are two main uses of tags:

1. **General management of resources**: Tags enable you to categorize your AWS resources in different ways, such as by purpose, owner, or environment. This can simplify filtering and searching across different resources. You can also use resource groups to create a custom console that organizes and consolidates the information you need based on your project and the resources you use. You can also create a resource group to view resources from different regions on the same screen, as shown in Figure 2.

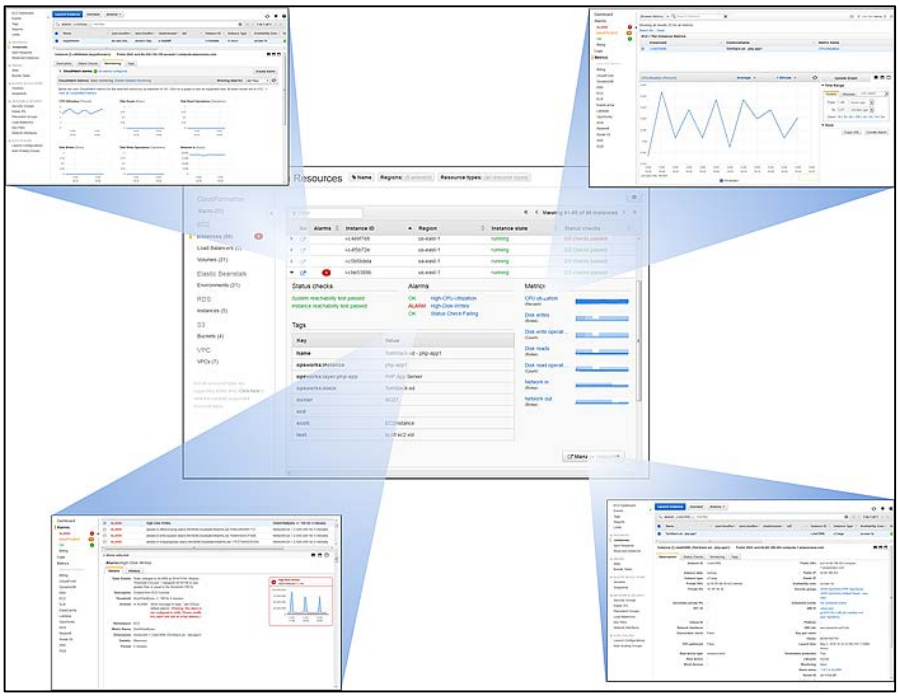

Figure 2: AWS Resource Groups

2. **Billing segregation**: Tags enable cost allocation reports and allow you to get cost segregation based on a particular business unit or environment, depending on the tagging strategy used. This along with AWS Cost Explorer can greatly simplify the billing data related visibility & reports.

Chargeback Module

Another important aspect of a multi-tenant system is cost segregation across tenants based on their usage. From an AWS resources perspective, tagging can be a great resource to help you separate out usage at a macro level. However, for most SaaS solutions, greater controls are needed for usage monitoring, so we recommend that you build your own custom billing module as needed.

A billing module could look like the high-level, generic example shown in Figure 3.

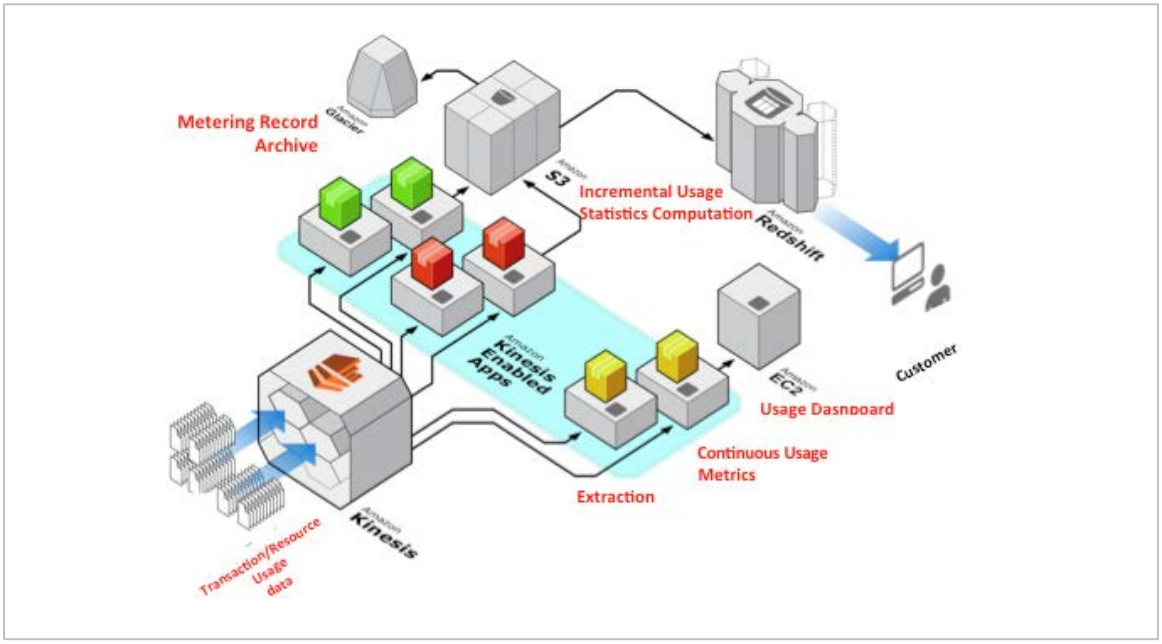

Figure 3: Sample Metering and Chargeback Module

- All of the resources that are launched, stopped, and terminated are tracked, and the data is then sent to an Amazon Kinesis stream.
- Granular measurements, such as the number of API requests made or the time taken to process any request, are tracked and the data is then fed into the Kinesis stream in real time.
- Two types of consumer applications can process the data stored in Amazon Kinesis:
  - A consumer fleet that generates real-time metrics on how the system is being utilized by various tenants. This may help you make decisions such as whether to throttle a particular tenant's usage, or perform other corrective actions based on real-time feeds.
  - A second set of a Kinesis consumer fleet could aggregate the continuous feed and generate monthly or quarterly usage reports for billing. It could also provide usage analytics for each tenant by processing the raw data and storing it in Amazon Redshift. For historical data processing or transformation, Amazon EMR can be used.

SaaS Solutions – Tenant Isolation Architecture Patterns

There are multiple approaches to deploying a packaged solution on AWS, ranging from a fully isolated deployment to a completely shared SaaS-type architecture, with many other deployment options in between. In order to support any of the deployment options, the solution or application itself should be able to support that SaaS multi-tenancy model, which is the basic assumption we will take here before diving deep into AWS-specific components of different deployment models.

The decision to pick a particular AWS deployment model depends on multiple criteria, including:

- Level of segregation across tenants and deployments
- Application scalability aspects across tenant-specific stacks
- Level of tenant-specific application customizations
- Cost of deployment
- Operations and management efforts
- End-tenant metering and billing aspects

The different choices are a "Rubik's cube" of options that impact one another in potentially unforeseen ways. The goal of this paper is to help with these multi-dimensional, unforeseen impacts. The following sections describe some of the SaaS deployment models on AWS, and include a pros and cons section for each option, to help guide you to the optimal solution given your business and technical requirements, as below:

- Model #1 – Tenant Isolation at the AWS Account Layer
- Model #2 – Tenant Isolation at the Amazon VPC Layer
- Model #3 – Tenant Isolation at Amazon VPC Subnet Layer
- Model #4 – Tenant Isolation at the Container Layer
- Model #5 – Tenant Isolation at the Application Layer

Model # 1 – Tenant Isolation at the AWS Account Layer

In this model, all the tenants will have their individual AWS accounts and will be isolated to an extent. In essence, this is not truly a multi-tenant SaaS solution, but can be treated as a managed solution on AWS.

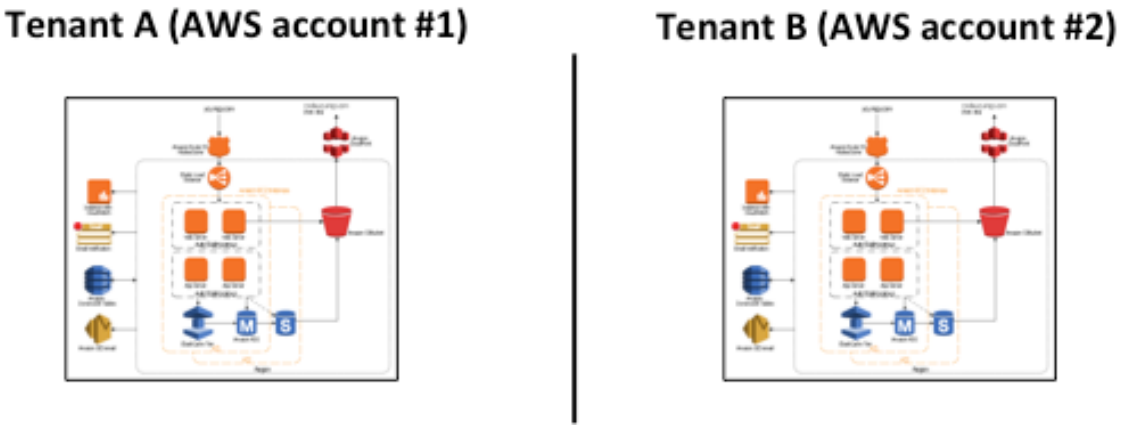

Figure 4: Tenant Isolation at AWS Account Layer

Pros:

- Tenants are completely separated out, and they do not have any overlap, which can provide each tenant with a greater sense of security.
- Solution or general configuration customizations are easy, because every deployment is specific to a tenant (or organization).
- It's easy to track AWS usage, because a separate monthly bill is generated for each tenant (or organization).

Cons:

- This option lacks the resources and cost optimizations that can be achieved by the economies of scale provided by a multi-tenant SaaS model.
- With a large number of tenants, it can become challenging to manage separate AWS accounts and individual tenant deployments from an operations perspective.
- As a best practice, all the AWS account root logins should be multi-factor authentication (MFA) enabled. With ever-increasing individual tenant accounts, it becomes difficult to manage all the MFA devices.

Best Practices:

- Centralized operations and management – IAM supports delegating access across AWS accounts for accounts you own using IAM roles. Using this functionality, you can manage all tenants' AWS accounts through your own common AWS account by assuming roles to perform various actions (such as launching a new stack using AWS CloudFormation or updating a security group configuration), instead of having to log in to each AWS account individually. You can utilize this functionality by using the AWS Management Console, AWS Command Line Interface (AWS CLI), and the API. Figure 3 provides a snapshot of how to set this up from the AWS Management Console.

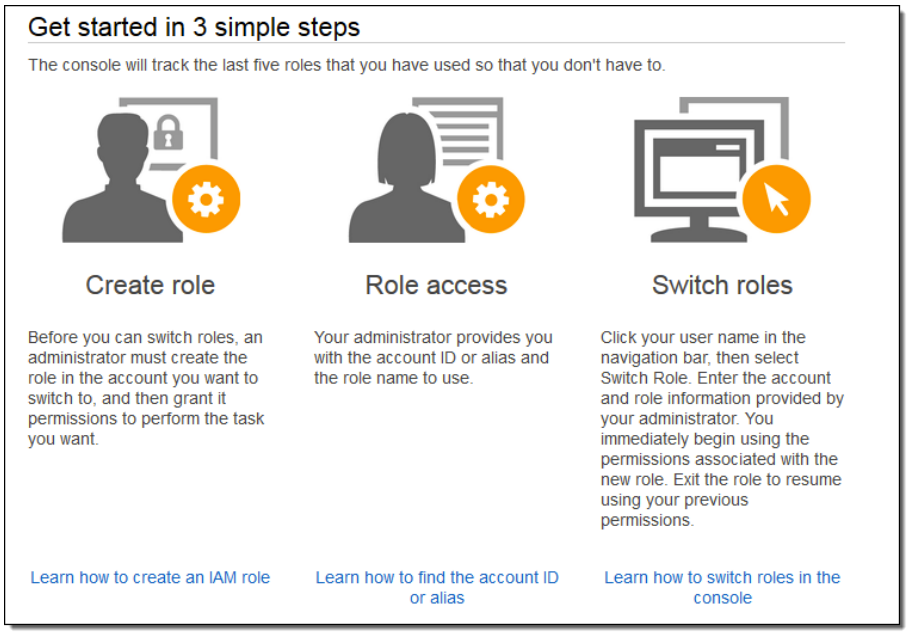

Figure 5: Cross-Account, IAM Role-based Access Setup

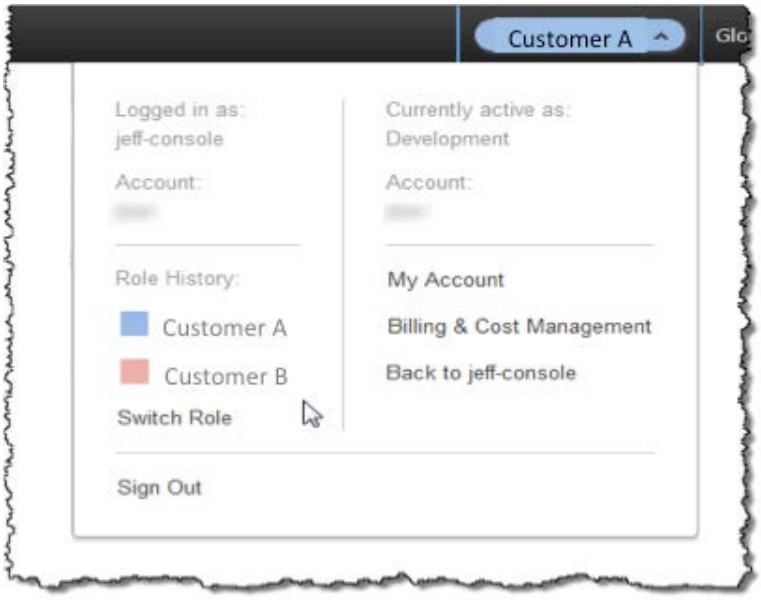

Figure 6: Cross-Account, IAM Role-based Access Switching

- Consolidated AWS billing – You can use the Consolidated Billing feature to consolidate payment for multiple AWS accounts within your organization by designating one of them to be the payer account. With Consolidated Billing, you can see a combined view of AWS charges incurred by all accounts, and you can get a detailed cost report for each individual AWS account associated with your payer account.

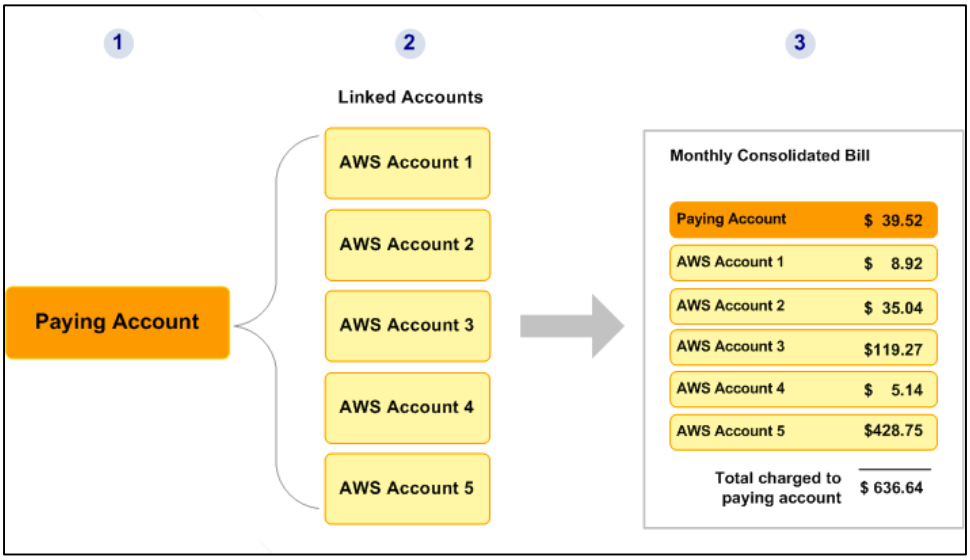

Figure 7: AWS Consolidated Billing

- VPC peering – If you would like to have a central set of services (say, for backup, anti-virus, OS patching, and so on), you can use a VPC peering connection in the same AWS region between your common AWS account that has these shared services and the respective tenant's AWS account. However, note that you are charged for data transfer within a VPC peering connection at the same rate as data transfer across Availability Zones. Therefore, you should factor this cost into the solution's overall cost modeling exercise.

Model # 2 – Tenant Isolation at the Amazon VPC Layer

In this model, all the tenant solution deployments are in the same AWS account, but the level of separation is at the VPC layer. For every tenant deployment, there's a separate VPC, which provides logical separation between tenants.

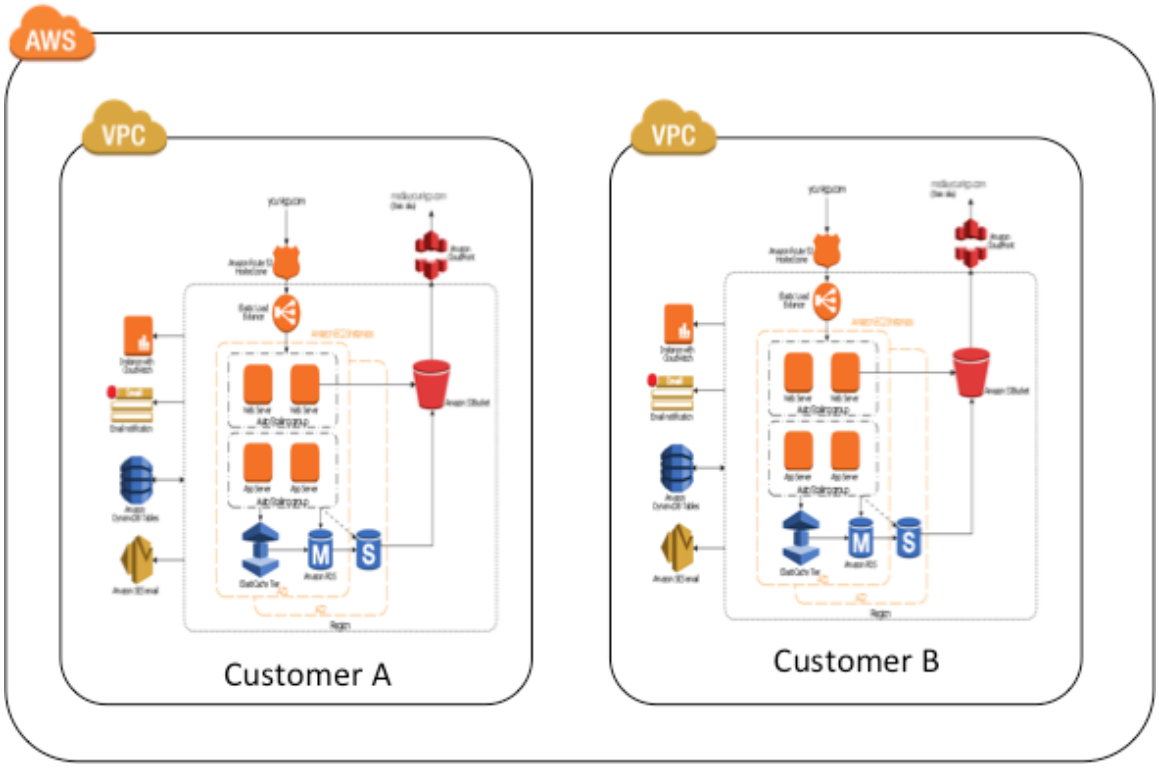

Figure 8: Tenant Isolation at VPC Layer

Pros:

- Everything is in a single account, so this model is easier to manage than a multi-account setup.
- There's appropriate isolation between different tenants, because each one lives in a different VPC.
- Compared with the previous model, this model provides better economies of scale and improved utilization of Amazon EC2 Reserved Instances, because all reservations and volume pricing constructs are applicable on the same AWS account. However, if Consolidated Billing is used, this model provides no advantage over the previous model, because Consolidated Billing treats all the accounts on the consolidated bill as one account.

Cons:

- Amazon VPC-related limits will have to be closely monitored, both from an overall account perspective and from each tenant's VPC perspective.
- If all the VPCs need connectivity back to an on-premises setup, then managing individual VPN connections may become a challenge.
- Even though it's the same account, if a shared set of services needs to be provided (such as backups, anti-virus updates, OS updates, and so forth), then VPC peering will need to be set up from the shared services VPC to all tenant VPCs.
- Security groups are tied to a VPC, so depending on the deployment architecture, you may have to create and manage multiple security groups for each VPC.
- AWS supports tagging as described in the Amazon EC2 documentation. However, if you need to separate usage and costs for services and resources beyond the available tagging support, you should either build a custom chargeback layer, or have a separate AWS account strategy to help clearly demarcate individual tenant usage.

Best Practices:

In this setup, use tags to separate out AWS costs for each of the tenant deployments. You can define resource groups and manage tags there, instead of managing them at the individual resource level. Once you have defined the tagging strategy, you can use the monthly cost allocation reports to view a breakup of AWS costs by tags and segregate it as per your needs (see the sample report in Figure 9).

Total Cost	user:Owner	user:Stack	user:Cost Center	user:Application
0.95	DbAdmin	Test	80432	Widget2
0.01	DbAdmin	Test	80432	Widget2
3.84	DbAdmin	Prod	80432	Widget2
6.00	DbAdmin	Test	78925	Widget1
234.63	SysEng	Prod	78925	Widget1
0.73	DbAdmin	Test	78925	Widget1
0.00	DbAdmin	Prod	80432	Portal
2.47	DbAdmin	Prod	78925	Portal

Figure 9: Sample Cost Allocation Report

Model # 3 – Tenant Isolation at Amazon VPC Subnet Layer

In this model, we will discuss the case where we have a single AWS account and a single VPC for all tenant deployments. The isolation happens at the level of subnets, and each tenant has their own separate version of an application or solution with no sharing across tenants. Figure 10 illustrates this type of deployment.

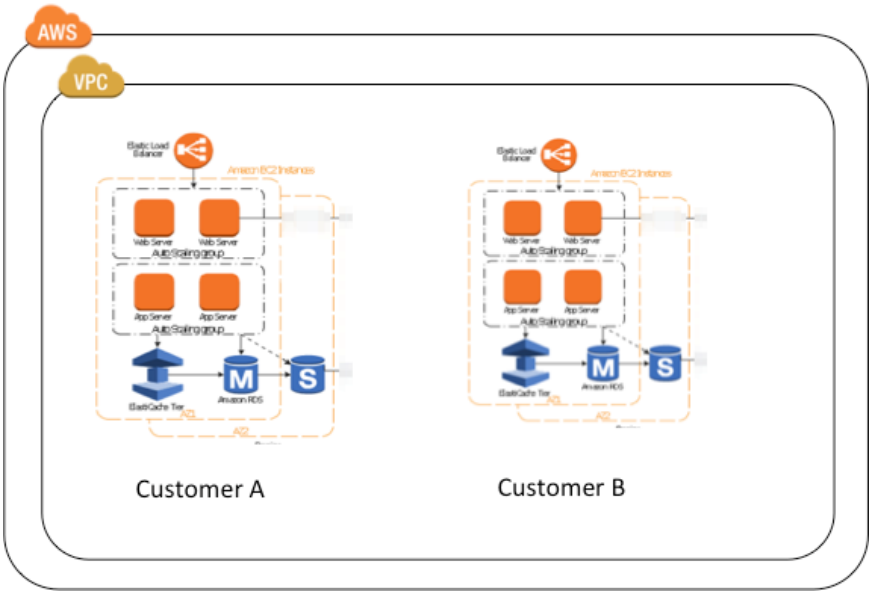

Figure 10: Tenant Isolation at VPC Subnet Layer

Pros:

- You don't need to set up VPC peering for intercommunication.
- VPN and AWS Direct Connect connectivity to a single on-premises site is simplified, as there is a single VPC.

Cons:

- Isolation between tenants has to be managed at the subnet level, so Amazon VPC network access control lists (NACLs) and security groups need to be carefully managed.
- VPC limits are harder to manage as the number of tenants increases. Furthermore, you can provision only a few subnets under the VPC CIDR (Classless Inter-Domain Routing), depending on its size, and the CIDR cannot be resized once created.
- Changing a VPC level setting (say, DHCP options set) affects all tenants although they have their individual deployments.
- There are limits on the number of security groups and the number of rules per security group at the VPC level, so managing those limits with multiple tenants in the same VPC may be complicated.

Best Practices:

- To access public AWS service endpoints (like Amazon S3), utilize VPC endpoints. This will scale better than routing the traffic for multiple tenants through a network address translation (NAT) instance.
- To avoid hitting security group-related limits in a VPC:
  - Consolidate security groups to stay under the limit.
  - Don't use security group cross-references; instead, refer to CIDR ranges.

Model # 4 – Tenant Isolation at the Container Layer

With the advent of container-based deployment, it is now possible to have a single instance and slice it for multiple tenant applications based on requirements. The Amazon EC2 Container Service (Amazon ECS) helps easily set up and manage Docker container-based deployments and could be used to deploy tenant-specific solution components in individual containers. Figure 11 illustrates a scenario where different tenants' containers are deployed in the same VPC.

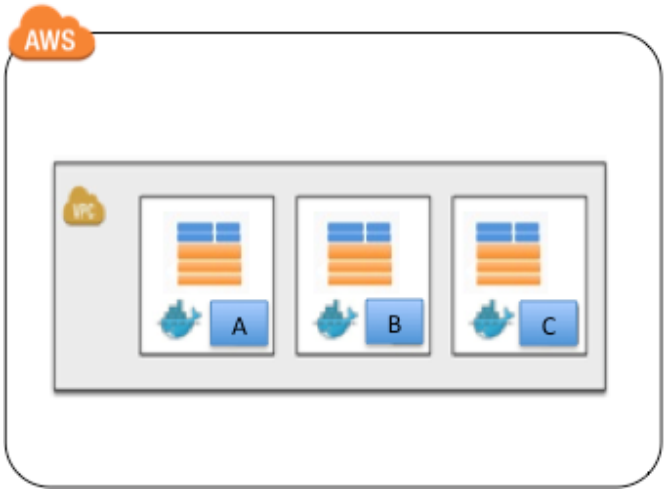

Figure 11: Tenant Isolation at Container Layer

Pros:

- You can have a higher level of resource utilization by having a container-based model on shared instances.
- It's easier to manage the clusters at scale, as Amazon ECS takes away the heavy lifting involved in terms of cluster management and general fault tolerance.
- Simplified deployments are possible, by testing a Docker image on any test/development environment and then using simple CLI-based options to directly put it into production.
- Amazon ECS deploys images on your own Amazon EC2 instances, which can be further segmented and controlled using VPC-based controls. This, along with Docker's own isolation model, meets the security requirements of most multi-tenant applications.

Cons:

- You can use Amazon EC2 and VPC security groups to limit the traffic on an Amazon EC2 instance. However, you need to manage the container configuration to control which ports are open. Managing those aspects may become a little tedious at scale.
- Tags do not work at the Amazon ECS task (container) level, so separating costs based on tags will not work, and a custom billing layer will be needed.

Best Practices:

- To secure container communication beyond the controls provided by VPC security groups, you could create a software-defined network for the containers, using point-to-point tunneling with Generic Routing Encapsulation (GRE) to route traffic between the container-based subnets.
- In order to architect auto scaling functionality using Amazon ECS, use a combination of Amazon CloudWatch and AWS Lambda-based container deployment. In this setup, an AWS Lambda function is triggered by an Amazon CloudWatch alarm to automatically add another Amazon ECS task to dynamically scale, as shown in Figure 12.

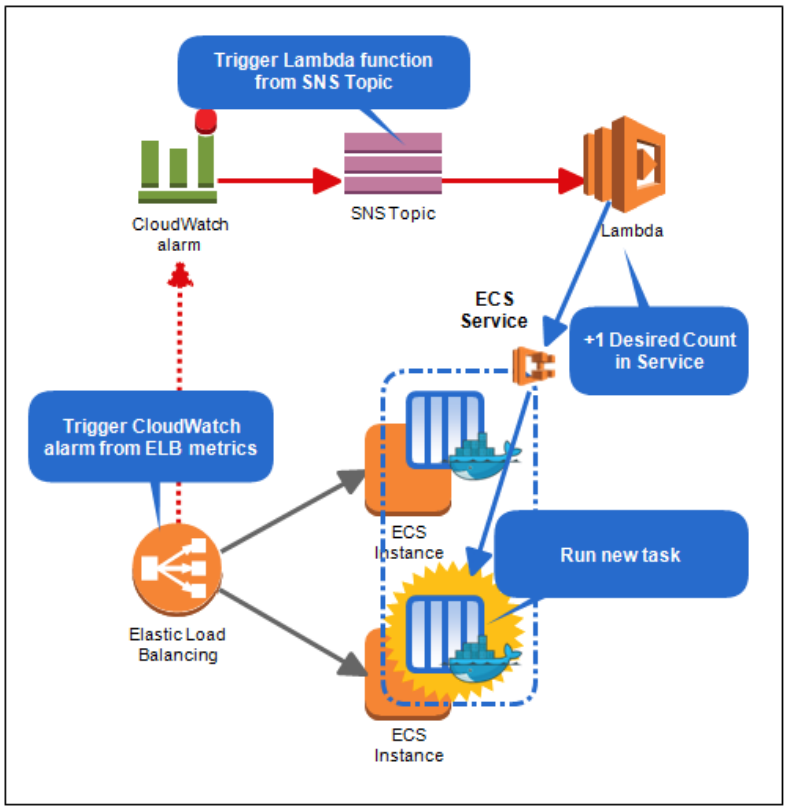

Figure 12: Auto scaling Architecture for Container-based Deployment

Model # 5 – Tenant Isolation at the Application Layer

This model represents a major shift from the earlier discussed models; now the application or solution deployment is shared across different tenants. This is a radical change and a movement toward a true multi-tenant SaaS model. However, to achieve this model, the application itself should be designed to support multi-tenancy. For example, if we take a typical 3-tier application with shared web and application layers, there can be some subtle variations at the database layer (which, for example, could be either Amazon RDS or a database on an Amazon EC2 instance):

a) **Separate databases**: Each tenant will have a different database for maximum isolation. To enable the application layers to pick up the right database upon each tenant's request, you will need to maintain metadata in a separate store (such as Amazon DynamoDB) where mapping of a tenant to its database is managed.
b) **Separate tables/schemas**: Different database flavors have different constructs, but another possible deployment model could be that all tenants' data resides in the same database, but the data is tied to different schemas or tables to provide a level of isolation.
c) **Shared database, shared schema/tables**: In this model, all tenants' data is placed together. A unique tenant ID column separates data records for each tenant. Whenever a new tenant needs to be added to the system, a new tenant ID is generated, additional capacity is provisioned, and traffic routing is started to an existing or new stack.

Pros:

- You can achieve economies of scale and better resource usage and optimization across the entire stack. As a result, this can often be the cheapest option to operate at scale when you have shared components across the architecture.
    - For example, having a huge multi-tenant Amazon DynamoDB table that can absorb the request spikes can be much cheaper than having higher provisioned Amazon DynamoDB tables for individual tenants.
- It's easy to manage and operate the stack, because it is a single deployment. Any changes or enhancements that need to be made are rolled out at once, rather than having to manage *n* different environments.
- Network connectivity is simplified, and the challenges around the VPC limits with other models are also subdued, because it's a single VPC deployment (although it may be bigger in size).

- All shared services (such as patching, OS updates, and anti-virus) are also centralized and deployed as a single unit for all the tenants.

Cons:

- Applications need to be multi-tenant aware, so existing applications may have to be re-architected.
- Depending on certain compliance and security requirements, co-hosting tenants with different security profiles may not be possible.

Best Practices:

To implement this model successfully, consider the following important aspects:

- Often times, different tenants have their own specific needs for certain features or customizations:
  - Try to group tenants according to their requirements; tenants with similar needs should be put on the same deployment.
  - Try to build the most asked for features in the core platform or application itself, and avoid customizations at the tenant level for long-term maintainability.
- Closely monitor the stack for each tenant's activities. If necessary, you should be able to throttle or deprioritize any particular tenant's actions to avoid affecting other tenants adversely.
- Ensure that you have the ability to scale the stacks up and down automatically, to address the changing needs of the tenants on a particular stack. This should be built into the architecture, rather than being done by manual updates.
- Use role-based and fine-grained access controls to enable access to limit a tenant's access across the entire stack. Amazon DynamoDB provides fine-grained access controls, which enable you to determine who can access individual data items and attributes in Amazon DynamoDB tables and indexes, and the actions that can be performed on them. Using Amazon DynamoDB in SaaS architectures can greatly reduce complexities.
- Another important aspect to handle is the AWS cost management across tenants according to their usage. To handle this, we recommend that you design a custom billing layer (as explained and outlined in previous sections) and incorporate it in the solution.

General Recommendations

Consider the following general best practices for a packaged SaaS solution design and delivery on AWS:

- Instead of building large, monolithic application architectures, it's often helpful to create smaller, independent, single-responsibility services that can be clubbed together to achieve the overall business functionality. These smaller microservices-based architectures can be easier to manage, and can independently scale. You could use services like Amazon ECS and AWS Lambda to create these smaller components. AWS Simple Queue Service (Amazon SQS) could also potentially help decouple microservices by introducing a queuing layer in between for communication. You can also use Amazon API Gateway to enable API-based interactions between the layers, thereby keeping them integrated just at the interface layer. To learn more about this microservices-based architecture pattern, see the blog post SquirrelBin: A Serverless Microservice Using AWS Lambda.
- Build abstraction at each layer so that you can future-proof your solution, by being able to change the underlying implementation without affecting the public interfaces. Consider aspects such as where you want the solution to be in next few years, and think about technology trends. For example, mobile was not as big five years ago as it is today. Plan for the future, and design your solution in a manner that is scalable and extensible to meet future needs.
- Define a release management process to enable frequent quality updates to the solution. AWS CodeCommit, AWS CodePipeline, and AWS CodeDeploy can help with this aspect of your deployment.
- Keep tenant-specific customizations to a minimum, and try to build most of the features within the platform itself. For tenant-specific configuration metadata, AWS DynamoDB can be useful.
- Build an API for your solution or platform if it needs to integrate with third-party systems.
- Use IAM roles for Amazon EC2, instead of using hard-coded credentials within various application components.
- Find ways to cost-optimize your solution. For instance, you can use Reserved or Spot Instances, adopt AWS Lambda to design an event-driven architecture, or use Amazon ECS to containerize smaller functional blocks.
- Utilize Auto Scaling to dynamically scale your environment up and down, as per load.
- Benchmark application performance to right-size your Amazon EC2 instances and their count.
- Make use of AWS Trusted Advisor recommendations to further optimize your AWS deployment.

- There are often custom capabilities that you may like to build into your platform that could be supplied by a packaged solution from an APN Technology Partner. Look for opportunities to pick and choose what to build on your own, versus utilizing an existing solution. Leverage various APN Partner solutions and offerings, and AWS Marketplace to augment the features and functionalities provided by AWS services.
- Enroll in the AWS SaaS Partner Program to learn, build, and grow your SaaS business on AWS.
- It's important to ensure that your solution can be effectively managed on AWS by your firm. Another option is to work with an AWS MSP Consulting Partner.
- Validate your operational model using the AWS operational checklist.
- Validate your security model using the AWS auditing security checklist.
- Leverage various APN Partner solutions and offerings, and AWS Marketplace to augment the features and functionalities provided by AWS services.

Conclusion

Every packaged SaaS solution is different in nature, but they share common ingredients. You can use the practices and architecture methodologies described in this paper to deploy a scalable, secure, optimized SaaS solution on AWS. The paper describes different models you can adopt. Depending on the type of SaaS solution you're building, using multiple models or even a hybrid approach may suit your needs.

Contributors

The following individuals and organizations contributed to this document:

- Kamal Arora, Solutions Architect, Amazon Web Services
- Tom Laszewski, Sr. Manager, Solutions Architects, Amazon Web Services
- Matt Yanchyshyn, Sr. Manager, Solutions Architects, Amazon Web Services

Further Reading

APN Partner Solutions

In order to build out various functions in a custom SaaS solution, you will likely want to integrate with popular ISV solutions across various functions. To make your selection easy, the APN has developed the AWS Competency Program, designed to highlight APN Partners who have demonstrated technical proficiency and proven customer success in specialized solution areas. Below are some of the AWS Competency solution pages, which you can refer to for more details:

- DevOps – https://aws.amazon.com/solutions/partners/dev-ops/
- Mobile - https://aws.amazon.com/mobile/partner-solutions/
- Security - https://aws.amazon.com/security/partner-solutions/
- Digital Media - https://aws.amazon.com/partners/competencies/digital-media/
- Marketing & Commerce - https://aws.amazon.com/digital-marketing/partner-solutions/
- Big Data - https://aws.amazon.com/partners/competencies/big-data/
- Storage - https://aws.amazon.com/backup-recovery/partner-solutions/

- Healthcare - https://aws.amazon.com/partners/competencies/healthcare/
- Life Sciences - https://aws.amazon.com/partners/competencies/life-sciences/
- Microsoft Solutions - https://aws.amazon.com/partners/competencies/microsoft/
- SAP Solutions - https://aws.amazon.com/partners/competencies/sap/
- Oracle Solutions - https://aws.amazon.com/partners/competencies/oracle/

Notes

- Details on various AWS usage and billing reports: http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/billing-what-is.html
- Amazon EC2 IAM roles: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
- Auto scaling Amazon ECS services using Amazon CloudWatch and AWS Lambda https://aws.amazon.com/blogs/compute/scaling-amazon-ecs-services-automatically-using-amazon-cloudwatch-and-aws-lambda/
- Working with Tag Editor- http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/tag-editor.html
- Working with resource groups: http://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/resource-groups.html
- Backup, archive, and restore approaches on AWS – http://d0.awsstatic.com/whitepapers/Backup_Archive_and_Restore_Approaches_Using_AWS.pdf
- AWS Managed Service Program - http://aws.amazon.com/partners/managed-service/
- AWS SaaS Partner program – http://aws.amazon.com/partners/saas/