AWS Whitepaper

Development and Test on Amazon Web Services

Development and Test on Amazon Web Services: AWS Whitepaper

Table of Contents

This whitepaper is for historical reference only. Some content might be outdated and some links might not be available.

Development and Test on Amazon Web Services

Publication date: **June 29, 2021** (*Document revisions*)

Abstract

This whitepaper describes how Amazon Web Services (AWS) adds value in the various phases of the software development cycle, with specific focus on development and test. For the development phase, this whitepaper:

- Shows you how to use AWS for managing version control
- Describes project management tools, the build process, and environments hosted on AWS
- Illustrates best practices

For the test phase, this whitepaper describes how to manage test environments and run various kinds of tests, including load testing, acceptance testing, fault tolerance testing, and so on.

AWS provides unique advantages in each of these scenarios and phases, enabling you to choose the ones most appropriate for your software development project. The intended audiences for this paper are project managers, developers, testers, systems architects, or anyone involved in software production activities.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

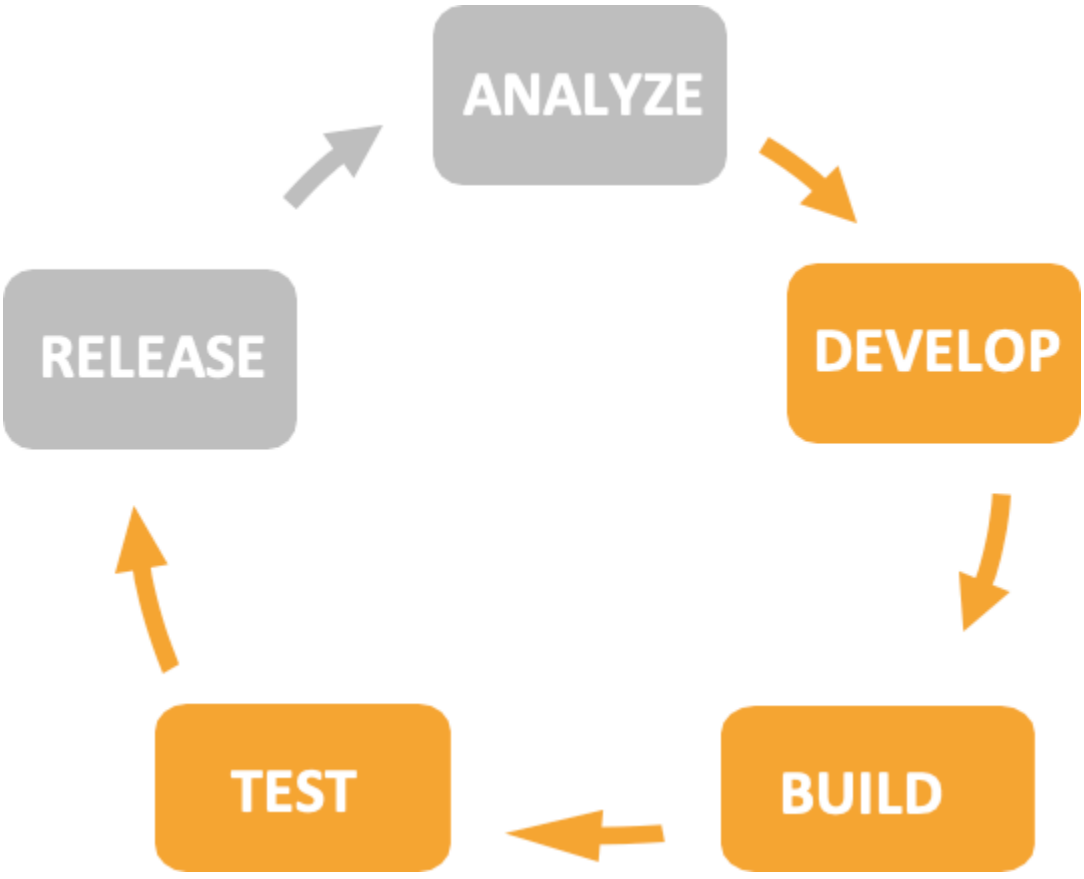
For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Introduction

Organizations write software for various reasons, ranging from core business needs (when the organization is a software vendor) to customizing or integrating software. Organizations also create different types of software: web applications, standalone applications, automated agents, and so on. In all such cases, development teams are pushed to deliver software of high quality as quickly as possible to reduce the time to market or time to production.

In this document, "development and test" refers to the various tools and practices applied when producing software. Regardless of the type of software to be developed, a proper set of development and test practices is key to success. However, producing applications not only requires software engineers, but also IT resources, which are subject to constraints like time, money, and expertise.

The software lifecycle typically consists of the following main elements:



Elements of the software lifecycle

This whitepaper covers aspects of the development, build, and test phases. For each of these phases, you need different types of IT infrastructure. AWS provides multiple benefits to software development teams. AWS offers on-demand access to a wide range of cloud infrastructure services, charging only for the resources that are used.

AWS helps eliminate both the need for costly hardware and the administrative pain that goes with owning and operating it.

Owning hardware and IT infrastructure usually involves a capital expenditure for a three-five year period, where most development and test teams need compute or storage for hours, days, weeks, or months. This difference in timescales can cause friction due to the difficulty for IT operations to satisfy simultaneous requests from project teams, even as they are constrained by a fixed set of resources. The result is that project teams spend a lot of time justifying, sourcing, and holding on to resources. This time could be spent focusing on the main job.

By provisioning only the resources needed for the duration of development phases, test runs, or complete test campaigns, your company can achieve important savings compared to investing up front in traditional hardware. With the right level of granularity, you can allocate resources depending on each project's needs and budget. In addition to those economic benefits, AWS also offers significant operational advantages, such as the ability to set up a development and test infrastructure in a matter of minutes rather than weeks or months, and to scale capacity up and down to provide the IT resources needed, only when they are needed.

This document highlights some of the best practices and recommendations around development and test on AWS. For example, for the development phase, this document discusses how to securely and durably set up tools and processes such as version control, collaboration environments, and automated build processes. For the testing phase, this document discusses how to set up test environments in an automated fashion, and how to run various types of tests, including side-by-side tests, load tests, stress tests, resilience tests, and more.

Development phase

Regardless of team size, software type being developed, or project duration, development tools are mandatory to rationalize the process, coordinate efforts, and centralize production. Like any IT system, development tools require proper administration and maintenance. Operating such tools on AWS not only relieves your development team from low-level system maintenance tasks such as network configuration, hardware setup, and so on, but also facilitates the completion of more complex tasks. The following sections describe how to operate the main components of development tools on AWS.

Source code repository

The source code repository is a key tool for development teams. As such, it needs to be available, and the data it contains (source files under version control) needs to be durably stored, with proper backup policies. Ensuring these two characteristics— availability and durability—requires resources, expertise, and time investment that typically aren't a core competency of a software development team.

Building a source code repository on AWS involves creating an AWS CodeCommit repository. AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. It eliminates the need for you to operate your own source control system, and there is no hardware to provision and scale or software to install, configure, and operate.

You can use CodeCommit to store anything from code to binaries, and it supports the standard functionality of GitHub, allowing it to work seamlessly with your existing GitHub-based tools. Your team can also use CodeCommit's online code tools to browse, edit, and collaborate on projects.

CodeCommit enables you to store any number of files, and there are no repository size limits. In a few simple steps, you can find information about a repository and clone it to your computer, creating a local repository where you can make changes and then push them to the CodeCommit repository. You can work from the command line on your local machines or use a GUI-based editor.

Project management tools

In addition to the source code repository, teams often use additional tools such as issue tracking, project tracking, code quality analysis, collaboration, content sharing, and so on. Most of the time, those tools are provided as web applications. Like any other classic web application, they require a server to run, and frequently a relational database. The web components can be installed

on Amazon Elastic Compute Cloud (Amazon EC2), with the database using Amazon Relational Database Service (Amazon RDS) for data storage.

Within minutes, you can create Amazon EC2 instances, which are virtual machines over which you have complete control. A variety of different operating systems and distributions are available as Amazon Machine Images (AMIs). An AMI is a template that contains a software configuration (operating system, application server, and applications) that you can run on Amazon EC2. After you've properly installed and configured the project management tool, AWS recommends you create an AMI from this setup so you can quickly recreate that instance without having to reinstall and reconfigure the software.

Project management tools have the same needs as source code repositories: they need to be available, and data has to be durably stored. While you can mitigate the loss of code analysis reports by recreating them against the desired repository version, losing project or issue tracking information might have more serious consequences. You can address the availability of the project management web application service by using AMIs to create replacement Amazon EC2 instances in case of failure.

You can store the application's data separately from the host system to simplify maintenance or migration operations. Amazon Elastic Block Store (Amazon EBS) provides off-instance storage volumes that persist independently from the life of an instance. After you create a volume, you can attach it to a running Amazon EC2 instance. As such, an Amazon EBS volume is provisioned and attached to the instance to store the data of the version control repository.

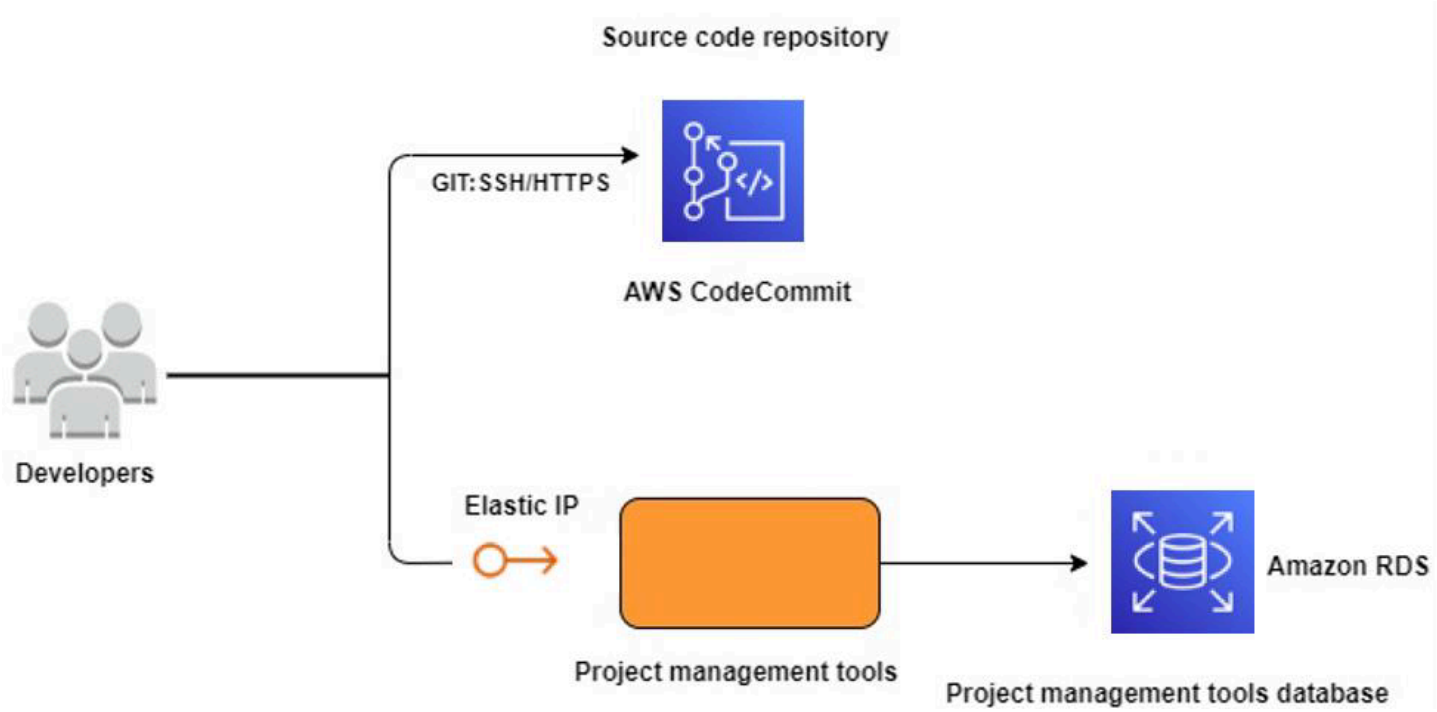
You achieve durability by taking point-in-time snapshots of the EBS volume containing the repository data. EBS snapshots are stored in Amazon Simple Storage Service (Amazon S3), a highly durable and scalable data store. Objects in Amazon S3 are redundantly stored on multiple devices across multiple facilities in an AWS Region. You can automate the creation and management of snapshots using Amazon Data Lifecycle Manager.

These snapshots can be used as the starting point for new Amazon EBS volumes, and can protect your data for long-term durability. In case of a failure, you can recreate the application data volume from the snapshots, and recreate the application instance from an AMI.

To facilitate proper durability and restoration, Amazon Relational Database Service (Amazon RDS) offers an easy way to set up, operate, and scale a relational database in AWS. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing the project team from this responsibility. Amazon RDS Database instances (DB instances) can be provisioned in a matter of minutes.

Optionally, Amazon RDS will ensure that the relational database software stays up to date with the latest patches. The automated backup feature of Amazon RDS enables point-in-time recovery for DB instances, allowing restoration of a DB instance to any point in time within the backup retention period.

An Elastic IP address provides a static endpoint to an Amazon EC2 instance, and can be used in combination with DNS (for example, behind a DNS CNAME). This helps teams to access their hosted services, such as the project management tool, in a consistent way, even if infrastructure is changed underneath; for example, when scaling up or down, or when a replacement instance is provisioned.



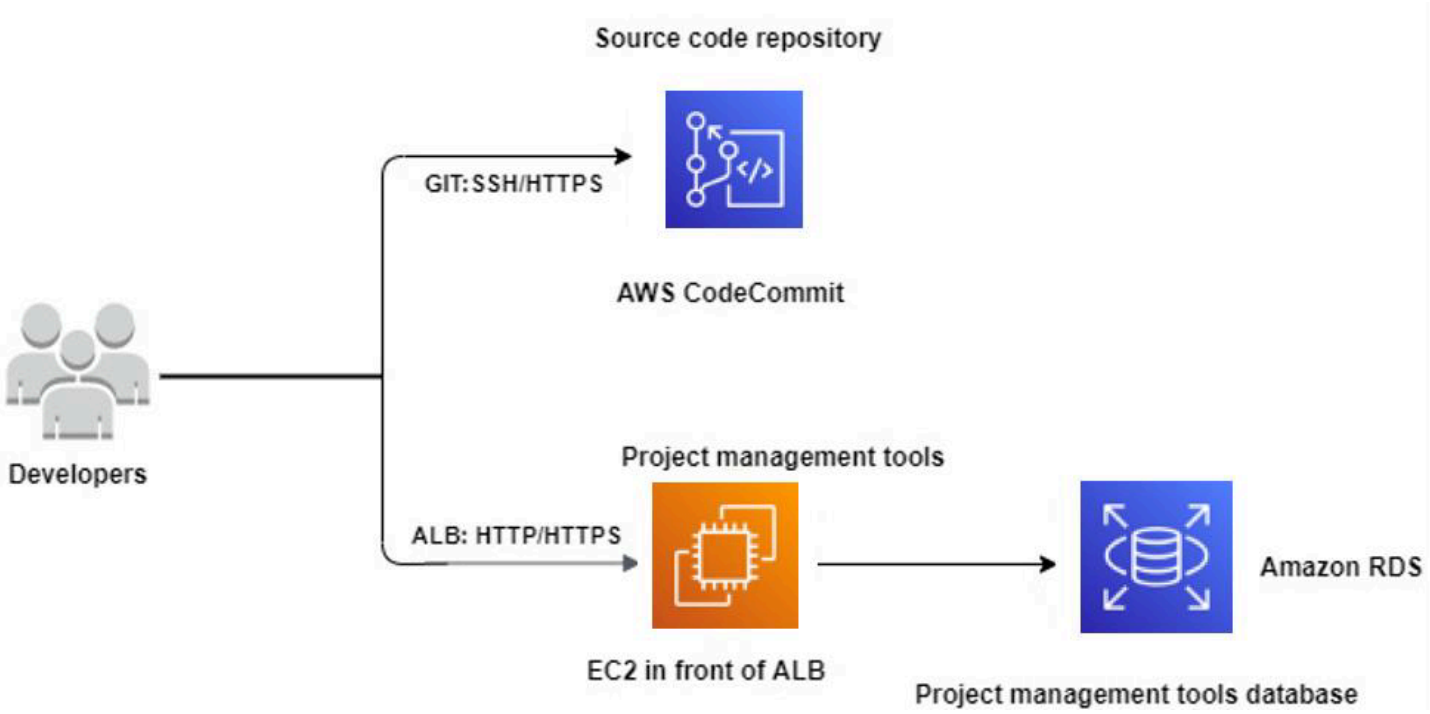
An elastic IP address provides a static endpoint to an Amazon EC2 instance

> **Note**
>
> : For even quicker and easier deployment, many project management tools are available from the AWS Marketplace or as Amazon Machine Images.

As your development team grows or adds more tools to the project management instance, you might require extra capacity for both the web application instance and the DB instance. In AWS,

scaling instances vertically is an easy and straightforward operation. You simply stop the EC2 instance, change the instance type, and start the instance.

Alternatively, you can create a new web application server from the AMI on a more powerful Amazon EC2 instance type, and replace the previous server. You can use horizontal scaling by using Elastic Load Balancing, adding more instances to the system by using AWS Auto Scaling. In this case, as you have more than one node, you can use Elastic Load Balancing to distribute the load across all application nodes. Amazon RDS DB instances can scale compute and memory resources with a few clicks on the AWS Management Console.



Use Elastic Load Balancing to distribute the load across all application nodes

When you want to quickly set up a software development project on AWS and don't want to configure custom project management tools on EC2, you can use AWS CodeStar. AWS CodeStar comes with a unified project dashboard and integration with Atlassian JIRA software, a third-party issue tracking and project management tool. With the AWS CodeStar project dashboard, you can easily track your entire software development process, from a backlog work item to production code deployment.

On-demand development environments

Developers primarily use their local laptops or desktops to run their development environments. This is typically where the integrated development environment (IDE) is installed, where unit tests are run, where source code is checked in, and so on.

However, there are a few cases where on-demand development environments hosted in AWS are helpful.

AWS Cloud9 is a cloud-based IDE that enables you to write, run, and debug your code with just a browser. It includes a code editor, debugger, and terminal. AWS Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, Ruby, Go, C++, and more, so you don't need to install files or configure your development machine to start new projects. Because your AWS Cloud9 IDE is cloud-based, you can work on your projects from your office, home, or anywhere using an internet-connected machine. With AWS Cloud9, you can quickly share your development environment with your team, enabling you to pair program and track each other's inputs in real time.

Some development projects may use specialized sets of tools that would be cumbersome or resource-intensive to install and maintain these on local machines, especially if the tools are used infrequently. For such cases, you can prepare and configure development environments with required tools (development tools, source control, unit test suites, IDEs, and so on), and then bundle them as AMIs.

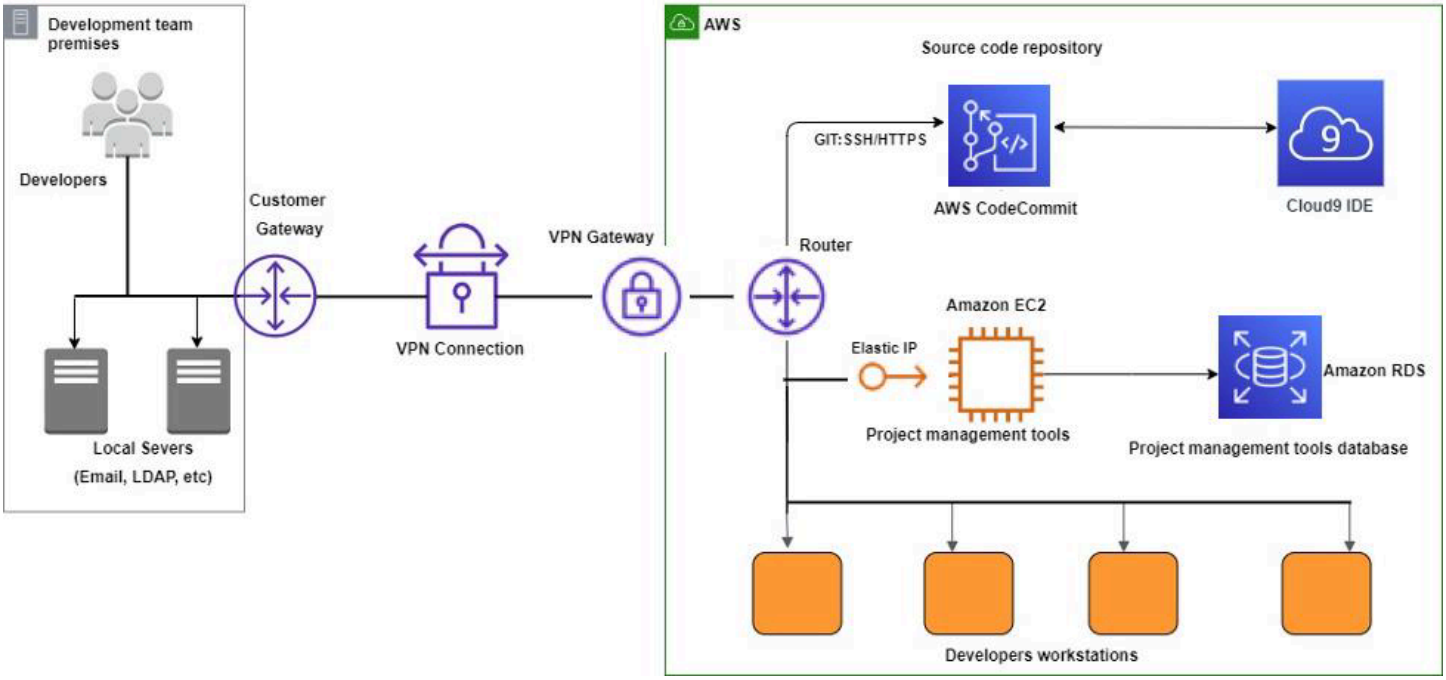
You can easily start the right environment and have it up and running in minimal time and with minimal effort. When you no longer need the environment, you can shut it down to free up resources. This can also be helpful if you need to switch context in the middle of having code checked out and work in progress. Instead of managing branches or dealing with partial check-ins, you can spin up a new temporary environment.

On AWS, you have access to a variety of different instance types, some with very specific hardware configurations. If you are developing specifically for a given configuration, it may be helpful to have a development environment on the same platform where the system is going to run. Amazon WorkSpaces enables you to provision virtual, cloud-based Microsoft Windows or Amazon Linux desktops for your users to run IDEs using your favorite applications such as Visual Studio, IntelliJ, Eclipse, AWS CLI, AWS SDK tools, Visual Studio Code, Eclipse, Atom, and many more.

The concept of hosted desktops is not limited to development environments; it can apply to other roles or functions as well. For more complex working environments, AWS CloudFormation makes

it easy to set up collections of AWS resources. This topic is discussed further in the *Testing* section of this document. In many cases, such environments are set up within the Amazon Virtual Private Cloud (Amazon VPC), which enables you to extend your on-premises private network to the cloud. You can then provision the development environments as if they were on the local network, but instead they are running in AWS. This can be helpful if such environments require any on-premises resources such as Lightweight Directory Access Protocol (LDAP).

The following diagram shows a deployment where development environments are running on Amazon EC2 instances within an Amazon VPC. Those instances are remotely accessed from an enterprise network, through a secure VPN connection.



Development environments running on Amazon EC2 instances within an Amazon VPC

Stopping vs. ending Amazon EC2 instances

Whenever development environments are not used; for example, during the hours when you are not working, or when a specific project is on hold, you can easily shut them down to save resources and cost. There are two possibilities:

- **Stopping** the instances, which is roughly equivalent to hibernating the operating system
- **Ending** the instances, which is roughly equivalent to discarding the operating system

When you stop an instance (possible for Amazon EBS–backed AMIs), the compute resources are released and no further hourly charges for the instance apply. The Amazon EBS volume stores the state, and next time you start the instance, it will have the working data as it did before you stopped it.

> **Note**
>
> : any data stored on ephemeral drives will not be available after a stop/start sequence.

When you end an instance, the root device and any other devices attached during the instance launch are automatically deleted (unless the DeleteOnTermination flag of a volume is set to "false"), meaning that data may be lost if there is no backup or snapshot available for the deleted volumes. An ended instance doesn't exist anymore and must be recreated from an AMI if needed. You would typically end the instance of a development environment if all work has been completed and/or the specific environment will not be used anymore.

If you use AWS Cloud9 IDE, the EC2 instance that AWS Cloud9 connects to by default stops 30 minutes after you close the IDE, and restarts automatically when you open the IDE. As a result, you typically only incur EC2 instance charges for when you are actively working.

If you chose to run your development environments on EC2 instances, you can use AWS Instance Scheduler to automatically stop your instances during weekends or non- working schedules. This can help reduce the instance utilization and overall spend.

Integrating with AWS APIs and IDE enhancements

With AWS, you can now code against and control IT infrastructure, either if the target platform of your project is AWS, or if the project is about orchestrating resources in AWS. For such cases, you can use the various AWS SDKs to easily integrate their applications with AWS APIs, taking the complexity out of coding directly against a web service interface and dealing with details around authentication, retries, error handling, and so on. The AWS SDK tools are available for multiple languages: C++, Go, JavaScript, Node.js, Python, Java, .Net, PHP, Ruby, and for mobile platforms Android and iOS.

AWS also offers IDE tools that make it easier for you to interact with AWS from within your IDEs, such as:

- AWS Toolkit for Visual Studio

- AWS Toolkit for VS Code
- AWS Toolkit for Eclipse
- AWS Toolkit for IntelliJ IDEA
- AWS Toolkit for PyCharm
- AWS Toolkit for Azure DevOps
- AWS Toolkit for Rider
- AWS Toolkit for WebStorm

For developing and building Serverless applications, AWS offers the Serverless Application Model (AWS SAM) open-source framework, which can be used with the AWS toolkits mentioned previously.

Build phase

The process of building an application involves many steps, including compilation, resource generation, and packaging. For large applications, each step involves multiple dependencies such as building internal libraries, using helper applications, generating resources in different formats, generating the documentation, and so on.

Some projects might require building the deliverables for multiple CPU architectures, platforms, or operating systems. The complete build process can take many hours, which has a direct impact on the agility of the software development team. This impact is even stronger on teams adopting approaches like continuous integration where every commit to the source repository triggers an automated build, followed by test suites.

Schedule builds

To mitigate this problem, teams working on projects with lengthy build times often adopt the "nightly build" (or neutral build) approach, or break the project into smaller sub- projects (or a combination of both). Doing nightly builds involves a build machine checking out the latest source code from the repository and building the project deliverables overnight. Development teams may not build as many versions as they would like, and the build should be completed in time for testing to begin the next day. Breaking down a project into smaller, more manageable parts might be a solution if each sub-project builds faster independently. However, an integration step combining all the different sub-projects is still often necessary for the team to keep an eye on the overall project, and to ensure the different parts still work well together.

On-demand builds

A more practical solution is to use more computational power for the build process. On traditional environments where the build server runs on hardware acquired by the organization, this option might not be viable due to economic constraints or provisioning delays. A build server running on an Amazon EC2 instance can be scaled up vertically in a matter of minutes, reducing build time by providing more CPU or memory capacity when needed.

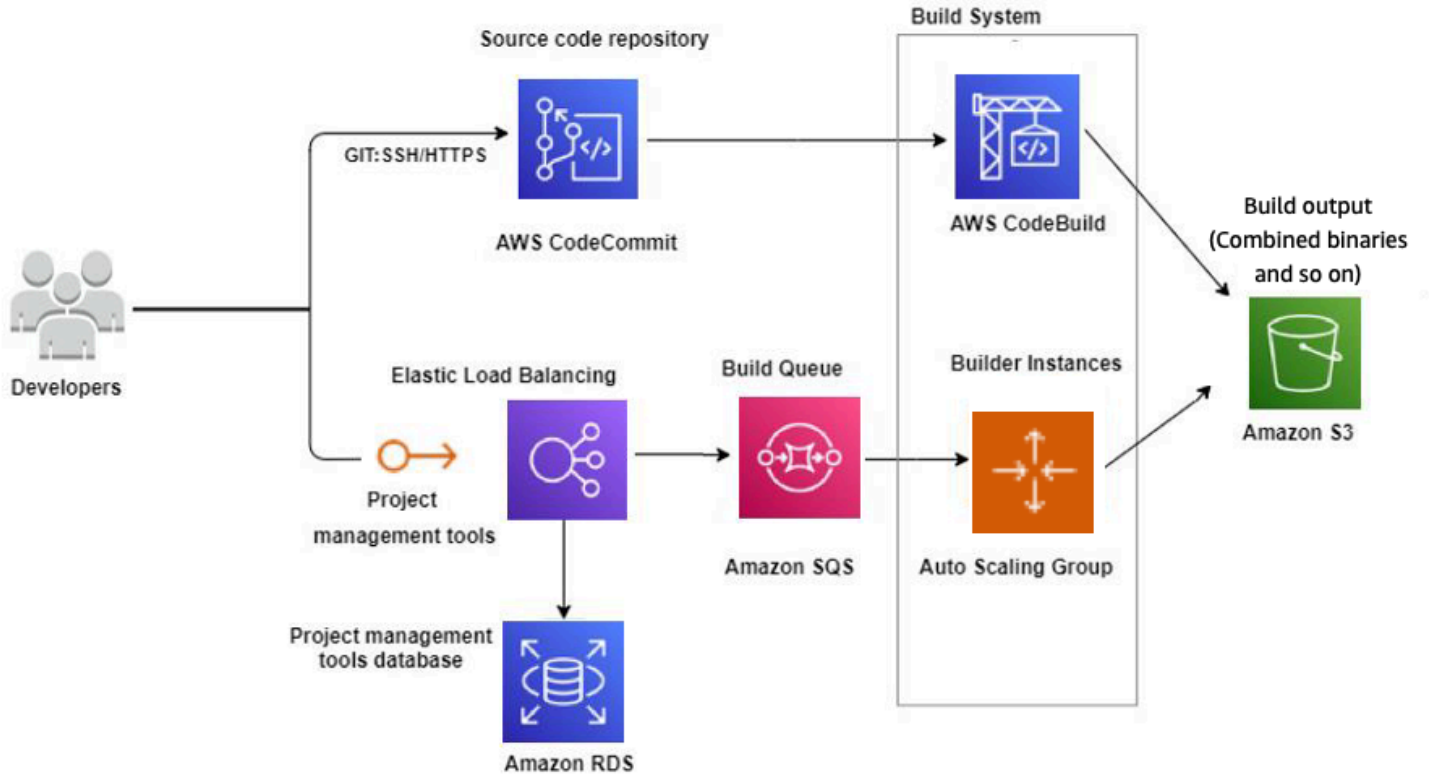
For teams with multiple builds triggered within the same day, a single Amazon EC2 instance might not be able to produce the builds quickly enough. A solution would be to take advantage of the on-demand and pay-as-you-go nature of AWS CodeBuild to run multiple builds in parallel. Every time

a new build is requested by the development team or triggered by a new commit to the source code repository, AWS CodeBuild creates a temporary compute container of the class defined in the build project and immediately processes each build as submitted. You can run separate builds concurrently without waiting in a queue. This also enables you to schedule automated builds at a specific time window.

If you use a build tool on EC2 instances running as a fleet of worker nodes, the task distribution to the worker nodes can be done using a queue holding all the builds to process. Worker nodes pick the next build to process as they are free. To implement this system, Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly scalable, hosted queue service. Amazon SQS makes it easy to create an automated build workflow, working in close conjunction with Amazon EC2 and other AWS infrastructure services. In this setup, developers commit code to the source code repository, which in turn pushes a build message into an Amazon SQS queue. The worker nodes poll this queue to pull a message and run the build locally according to the parameters contained in the message (for example, the branch or source version to use).

You can further enhance this setup by dynamically adjusting the pool of worker nodes consuming the queue. Auto Scaling is a service that makes it easy to scale the number of worker nodes up or down automatically according to predefined conditions. With Auto Scaling, worker nodes' capacity can increase seamlessly during demand spikes to maintain quick build generation, and decrease automatically during demand lulls to minimize costs.

You can define scaling conditions using Amazon CloudWatch, a monitoring service for AWS Cloud resources. For example, Amazon CloudWatch can monitor the number of messages in the build queue and notify Auto Scaling that more or less capacity is needed depending on the number of messages in the queue. The following diagram summarizes this scenario:



Amazon CloudWatch can monitor the number of messages in the build queue and notify Auto Scaling that more or less capacity is needed

Storing build artifacts

Every time you produce a build, you need to store the output somewhere. Amazon S3 is an appropriate service for this. Initially, the amount of data to be stored for a given project is small, but it grows over time as you produce more builds. Here the pay-as- you-go and capacity characteristics of S3 are particularly attractive. When you no longer need the build output, you can delete it, or use the S3 lifecycle policies to delete or archive the objects to Amazon Glacier storage class. AWS CodeBuild by default uses S3 buckets to store the build outputs.

To distribute the build output (for example, to be deployed in test, staging, or production, or to be downloaded to clients), AWS offers several options. You can distribute build output packages directly out of S3 by configuring bucket policies and/or ACLs to restrict the distribution. You can also share the output object using an S3 presigned URL.

Another option is to use Amazon CloudFront, a web service for content delivery, which makes it easy to distribute packages to end users with low latency and high data transfer speeds, thereby

improving the end user experience. This can be helpful, for example, when a large number of clients are downloading install packages or updates. Amazon CloudFront offers several options; for example, to authorize and/or restrict access, though a full discussion of this is out of scope for this document.

Testing phase

Tests are a critical part of software development. They ensure software quality, but more importantly, they help find issues early in the development phase, lowering the cost of fixing them later during the project. Tests come in many forms: unit tests, performance tests, user acceptance tests, integration tests, and so on, and all require IT resources to run. Test teams face the same challenges as development teams: the need for enough IT resources, but only during the limited duration of the test runs. Test environments change frequently and are different from project to project, and may require different IT infrastructure or have varying capacity needs.

The AWS on-demand and pay-as-you-go value propositions are well adapted to those constraints. AWS enables your test teams to eliminate both the need for costly hardware and the administrative pain that goes along with owning and operating it.

AWS also offers significant operational advantages for testers. Test environments can be set up in minutes rather than weeks or months, and a variety of resources, including different instance types, are available to run tests whenever they are needed.

Automating test environments

There are many software tools and frameworks available for automating the process of running tests, but proper infrastructure must be in place. This involves provisioning infrastructure resources, initializing the resources with a sample dataset, deploying the software to be tested, orchestrating the test runs, and collecting results. The challenge is not only to have enough resources to deploy the complete application with all the different servers or services it might require, but to be able to initialize the test environment with the right software and the right data over and over. Test environments should be identical between test runs; otherwise, it is more difficult to compare results.

Another important benefit of running tests on AWS is the ability to automate them in various ways. You can create and manage test environments programmatically using the AWS APIs, CLI tools, or AWS SDKs. Tasks that require human intervention in classic environments (allocating a new server, allocating and attaching storage, allocating a database, and so on) can be fully automated on AWS using AWS CodePipeline and AWS CloudFormation.

For testers, designing tests suites on AWS means being able to automate a test down to the operation of the components, which are traditionally static hardware devices.

Automation makes test teams more efficient by removing the effort of creating and initializing test environments, and less error prone by limiting human intervention during

the creation of those environments. An automated test environment can be linked to the build process, following continuous integration principles. Every time a successful build is produced, a test environment can be provisioned and automated tests run on it.

The following sections describe how to automatically provision Amazon EC2 instances, databases, and complete environments.

Provisioning instances

You can easily provision Amazon EC2 instances from AMIs. An AMI encapsulates the operating system and any other software or configuration files pre-installed on the instance. When you launch the instance, all the applications are already loaded from the AMI and ready to run. For information about creating AMIs, refer to the Amazon EC2 documentation. The challenge with AMI-based deployments is that each time you need to upgrade software, you have to create a new AMI. Although the process of creating a new AMI (and deleting an old one) can be completely automated using EC2 Image Builder, you must define a strategy for managing and maintaining multiple versions of AMIs.

An alternative approach is to include only components into the AMI that don't change often (operating system, language platform and low-level libraries, application server, and so on). More volatile components, like the application under development, can be fetched and deployed to the instance at runtime. For more details on how to create self- bootstrapped instances, refer to Bootstrapping.

Provisioning databases

Test databases can be efficiently implemented as Amazon RDS database instances. Your test teams can instantiate a fully operational database easily, and load a test dataset from a snapshot. To create this test dataset, you first provision an Amazon RDS instance. After injecting the dataset, you create a snapshot of the instance. From that time, every time you need a test database for a test environment, you can create one as an Amazon RDS instance from that initial snapshot. Refer to Restoring from a DB snapshot. Each Amazon RDS instance started from the same snapshot will contain the same dataset, which helps ensure that your tests are consistent.

Provisioning complete environments

While you can create complex test environments containing multiple instances using the AWS APIs, command line tools, or the AWS Management Console, AWS CloudFormation makes it even easier to create a collection of related AWS resources and provision them in an orderly and predictable fashion.

AWS CloudFormation uses templates to create and delete a collection of resources together as a single unit (a *stack*). A complete test environment running on AWS can be described in a template, which is a text file in JSON or YAML format. Because templates are just text files, you can edit and manage them in the same source code repository you use for your software development project. That way, the template will mirror the status of the project, and test environments matching older source versions can be easily provisioned. This is particularly useful when dealing with regression bugs. In just a few steps, you can provision the full test environment, enabling developers and testers to simulate a bug detected in older versions of the software.

AWS CloudFormation templates also support parameters that can be used to specify a specific software version to be loaded, the Amazon EC2 instance sizes for the test environment, the dataset to be used for the databases, and so on.

Provisioning cloud applications can be a challenging process that requires you to perform manual actions, write custom scripts, maintain templates, or learn domain- specific languages. You can now use the AWS Cloud Development Kit (AWS CDK) (AWS CDK), an open-source software development framework for defining cloud infrastructure-as-code with modern programming languages, and deploying it through AWS CloudFormation. AWS CDK uses familiar programming languages such as TypeScript, JavaScript, Python, Java, C# / .Net, and Go for modeling your applications.

For more information about how to create and automate deployments on AWS using AWS CloudFormation, refer to AWS CloudFormation Resources.

Load testing

Functionality tests running in controlled environments are valuable tools to ensure software quality, but they give little information on how an application or a complete deployment will perform under heavy load. For example, some websites are specifically created to provide a service for a limited time: ticket sales for sports events, special sales, limited edition launches, and so on. Such websites must be developed and architected to perform efficiently during peak usage periods.

In some cases, the project requirements clearly state the minimum performance metrics to be met under heavy load conditions (for example, search results must be returned in under 100 milliseconds (ms) for up to 10,000 concurrent requests), and load tests are exercised to ensure that the system can sustain the load within those limits.

For other cases, it is not possible or practical to specify the load a system should sustain. In such cases, load tests are performed to measure the behavior under heavy load conditions. The objective is to gradually increase the load of a system, to determine the point where the performance degrades in such a way that the system cannot operate anymore.

Load tests simulate heavy inputs that exercise and stress a system. Depending on the project, inputs can be a large number of concurrent incoming requests, a huge dataset to process, and so on. One of the main difficulties in load testing is generating large enough amounts of inputs to push the tested system to its limits. Typically, you need large amounts of IT resources to deploy the system to test, and to generate the test input, which requires further infrastructure. Because load tests generally don't run for more than a couple of hours, the AWS pay-as-you-go model nicely fits this use case.

You can also automate load tests using the techniques described in the previous section, enabling your testers to exercise them more frequently to ensure that each major change to the project doesn't adversely affect system performance and efficiency. Conversely, by launching automated load tests, you can discover whether a new algorithm, caching layer, or architecture design is more efficient and benefits the project.

> **Note**
>
> : For quick and easy setup, testing tools and solutions are also available from the AWS Marketplace.

In Serverless architectures using AWS services such as AWS Lambda, Amazon API Gateway, AWS Step Functions, and so on, load testing can help identify custom code in Lambda functions that may not run efficiently as traffic scales up. It also helps to determine an optimum timeout value by analyzing your functions' running duration to identify problems with a dependency service. One of the most popular tools to perform this task is Artillery Community Edition, which is an open-source tool for testing serverless APIs. You can also use Distributed Load Testing on AWS to automate application testing, understand how it performs at scale, and fix bottlenecks before releasing your application.

Network load testing

Testing an application or service for network load involves sending large numbers of requests to the system being tested. There are many software solutions available to simulate request scenarios, but using multiple Amazon EC2 instances may be necessary to generate enough traffic. Amazon EC2 instances are available on-demand and are charged by the hour, which makes them well suited for network load testing scenarios. Keep in mind the characteristics of different instance types. Generally, larger instance types provide more input/output (I/O) network capacity, the primary resource consumed during network load tests.

With AWS, test teams can also perform network load testing on applications that run outside of AWS. Having load test agents dispersed in different Regions of AWS enables testing from different geographies; for example, to get a better understanding of the end user experience. In that scenario, it makes sense to collect log information from the instances that simulate the load. Those logs contain important information such as response times from the tested system. By running the load agents from different Regions, the response time of the tested application can be measured for different geographies. This can help you understand the worldwide user experience.

Because you can end load-testing Amazon EC2 instances right after the test, you should transfer log data to S3 for storage and later analysis.

When you plan to run high volume network load tests directly from your EC2 instances to other EC2 instances, follow the Amazon EC2 Testing Policy.

Load testing for AWS

Load testing an application running on AWS is useful to make sure that elasticity features are correctly implemented. Testing a system for network load is important to make sure that for web front-ends, Auto Scaling, and Elastic Load Balancing configurations are correct. Auto Scaling offers many parameters and can use multiple conditions defined with Amazon CloudWatch to scale the number of front-end instances up or down.

These parameters and conditions influence how fast an Auto Scaling group will add or remove instances. An Amazon EC2 instance's post-provisioning time might also affect an application's ability to scale up quickly enough. After initialization of the operating system running on Amazon EC2 instances, additional services are initialized, such as web servers, application servers, memory caches, middleware services, and so on. The initialization time of these different services affects the scale-up delay, especially when additional software packages need to be pulled down from a

repository. Load testing provides valuable metrics on how fast additional capacity can be added into a particular system.

Auto Scaling is not only used for front-end systems. You might also use it for scaling internal groups of instances, such as consumers polling an Amazon SQS queue or workers and deciders participating in an Amazon Simple Workflow Service (Amazon SWF) workflow. In both cases, load testing the system can help ensure you've correctly implemented and configured Auto Scaling groups or other automated scaling techniques to make your final application as cost-effective and scalable as possible.

Cost optimization with Spot instances

Load testing can require many instances, especially when exercising systems that are designed to support a high amount of load. While you can provision Amazon EC2 instances on-demand and discard them when the test is completed while only paying by the hour, there is an even more cost-effective way to perform those tests using Amazon EC2 Spot Instances.

Spot Instances enable customers to bid for unused Amazon EC2 capacity. Instances are charged the Spot Price set by Amazon EC2, which fluctuates depending on the supply of and demand for Spot Instance capacity. To use Spot Instances, place a Spot Instance request specifying the instance type, the desired Availability Zone, the number of Spot Instances to run, and the maximum price to pay per instance hour.

The Spot Instance Price history for the past 90 days is available via the Amazon EC2 API or the AWS Management Console. If the maximum price bid exceeds the current Spot Price, the request is fulfilled and instances are started. The instances run until either they are ended or the Spot Price increases above the maximum price, whichever is sooner.

Refer to Testimonials and Case Studies to read about other customers' case studies and testimonials on EC2 Spot instances.

User acceptance testing

The objective of user acceptance testing is to present the current release to a testing team representing the final user base, to determine if the project requirements and specification are met. When users can test the software earlier, they can spot conceptual weaknesses that have been introduced during the analysis phase, or clarify gray areas in the project requirements.

By testing the software more frequently, users can identify functional implementation errors and user interface or application flow misconceptions earlier, lowering the cost and impact of

correcting them. Flaws detected by user acceptance testing may be very difficult to detect by other means. The more often you conduct acceptance tests, the better for the project, because end users provide valuable feedback to development teams as requirements evolve.

However, like any other test practice, acceptance tests require resources to run the environment where the application to be tested will be deployed. As described in previous sections, AWS provides on-demand capacity as needed in a cost-effective way, which is also appropriate for acceptance testing. Using some of the techniques described previously, AWS enables complete automation of the process of provisioning new test environments and of disposing environments no longer needed. Test environments can be provided for certain times only, or continuously from the latest source code version, or for every major release.

By deploying the acceptance test environment within Amazon VPC, internal users can transparently access the application to be tested. Such an application can also be integrated with other production services inside the company, such as LDAP, email servers, and so on, offering a test environment to the end users that is even closer to the real and final production environment.

Side-by-side testing

Side-by-side testing is a method used to compare a control system to a test system. The goal is to assess whether changes applied to the test system improve a desired metric compared to the control system. You can use this technique to optimize the performance of complex systems where a multitude of different parameters can potentially affect the overall efficiency. Knowing which parameter will have the desired effect is not always obvious, especially when multiple components are used together and influence the performance of each other.

You can also use this technique when introducing important changes to a project, such as new algorithms, caches, different database engines, or third-party software. In such cases, the objective is to ensure your changes positively affect the global performance of the system.

After you've deployed the test and control systems, send the same input to both, using load-testing techniques or simple test inputs. Finally, collect performance metrics and logs from both systems and compare them to determine if the changes you introduced in the test system present an improvement over the control system.

By provisioning complete test environments on-demand, you can perform side-by-side tests efficiently. While you can do side-by-side testing without automated environment provisioning, using the automation techniques described above makes it easier to perform those tests whenever

needed, taking advantage of the pay-as-you-go model of AWS. In contrast, with traditional hardware, it may not be possible to run multiple test environments for multiple projects simultaneously.

Side-by-side tests are also valuable from a cost optimization point of view. By comparing two environments in different AWS accounts, you can easily come up with cost and performance ratios to compare both environments. By continuously testing architecture changes for cost performance, you can optimize your architectures for efficiency.

Fault-tolerance testing

When AWS is the target production environment for the application you've developed, some specific test practices provide insights into how the system will handle corner cases, such as component failures. AWS offers many options for building fault-tolerant systems. Some services are inherently fault-tolerant, for example, Amazon S3, Amazon DynamoDB, Amazon SimpleDB, Amazon SQS, Amazon Route 53, Amazon CloudFront, and so on. Other services such as Amazon EC2, Amazon EBS, and Amazon RDS provide features that help architect fault-tolerant and highly available systems.

For example, Amazon RDS offers the Multi-Availability Zone option that enhances database availability by automatically provisioning and managing a replica in a different Availability Zone. For more information, see the resources available in the AWS Architecture Center.

Many AWS customers run mission-critical applications on AWS, and they need to make sure their architecture is fault tolerant. As a result, an important practice for all systems is to test their fault-tolerance capability. While a test scenario exercises the system (using similar techniques to load testing), some components are taken down on purpose to check if the system is able to recover from such simulated failure. You can use the AWS Management Console or the CLI to interact with the test environment.

For example, you might end Amazon EC2 instances, and then test whether an Auto Scaling group is working as expected and a replacement instance automatically provisioned. You can also automate this kind of test by integrating AWS Fault Injection Simulator with your CI/CD pipeline. It is a best practice is to use automated tools that, for example, occasionally and randomly disrupt Amazon EC2 instances. With Fault Injection Simulator, you can stress an application by creating disruptive events, such as a sudden increase in CPU or memory consumption, to observe how the system responds and implement improvements.

Resource management

With AWS, your development and test teams can have their own resources, scaled according to their own needs. Provisioning complex environments or platforms composed of multiple resources can be done using AWS CloudFormation stacks, or some of the other automation techniques described in this whitepaper. In large organizations comprising multiple teams, it is a good practice to create an internal role or service responsible for centralizing and managing IT resources running on AWS. This role typically consists of:

- Promoting the internal development and test practices described here
- Developing and maintaining template AMIs and template AWS CloudFormation stacks with the different tools and platforms used in your organization
- Collecting resource requests from project teams, and provisioning resources on AWS according to your organization's policies, including network configuration (such as Amazon VPC) and security configurations (such as Security Groups and IAM credentials)
- Monitoring resource usage and charges using AWS Cost Explorer, and allocating these to team budgets

You can use the Service Catalog to achieve the tasks above or you might want to develop your own internal provisioning and management portal for a tighter integration with internal processes. You can do this by using one of the AWS SDKs, which allow programmatic access to resources running on AWS.

Cost allocation and multiple AWS accounts

Some customers have found it helpful to create specific accounts for development and test activities. This can be important when your production environment also runs on AWS and you need to separate teams and responsibilities. Separate accounts are isolated from each other by default, so that, for example, development and test users do not interfere with production resources. To enable collaboration, AWS offers a number of features that enable sharing of resources across accounts, such as Amazon S3 objects, AMIs, and Amazon EBS snapshots.

To separate out and allocate the cost for the various activities and phases of the development and test cycle, AWS offers various options. One option is to use separate accounts (for example, for development, testing, staging, and production), and each account will have its own bill. You can

also consolidate multiple accounts using consolidated billing for AWS Organizations to simplify costs and take advantage of quantity discounts with a single bill.

Another option is to make use of the monthly cost allocation report, which enables you to organize and track your AWS costs by using resource tagging. In the context of development and test, tags can represent the various stages or teams of the development cycle, though you are free to choose the dimensions you find most helpful.

Conclusion

Development and test practices require certain resources at certain times for the development cycle. In traditional environments, those resources might not be available at all, or not in the necessary timeframe. When those resources are available, they provide a fixed amount of capacity that is either insufficient (especially in variable activities like testing), or wasted (but paid for) when the resources are not used. For more information, see the Auto Scaling documentation.

AWS offers a cost-effective alternative to traditional development and test infrastructures. Instead of waiting weeks or even months for hardware, you can instantly provision resources, scale up as the workload grows, and release resources when they are no longer needed.

Whether development and test environments consist of a few instances or hundreds, whether they are needed for a few hours or 24/7, you still pay only for what you use. AWS is a programming-language and operating system−agnostic platform, and you can choose the development platform or programming model used in your business. This flexibility enables you to focus on your project, not on operating and maintaining your infrastructure.

AWS also enables possibilities that are difficult to realize with traditional hardware. You can fully automate resources on AWS so that environments can be provisioned and decommissioned without human intervention. You can start development environments on-demand; kick off builds when needed, unconstrained by the availability of resources; provision test resources; and automatically orchestrate entire test runs or campaigns.

AWS offers you the ability to experiment and iterate with a rapidly changeable infrastructure. Your project teams are free to use inexpensive capacity to perform any kind of tests or to experiment with new ideas, with no upfront expenses or long-term commitments, making AWS a platform of choice for development and test.

Contributors

The following individuals and organizations contributed to this document:

- Rakesh Singh, Sr. Technical Account Manager, AMER-World Wide Public Sector
- Carlos Conde
- Attila Narin

Further reading

- Developer Tools on AWS
- *How AWS Pricing Works* (whitepaper)
- AWS Architecture Center
- AWS Whitepapers & Guides

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Whitepaper updated	Updated for latest technologies and services.	June 29, 2021
Initial publication	Whitepaper published.	November 2, 2012

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. 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

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.