



Development and Test on Amazon Web Services

November 2012
Carlos Conde, Attila Narin

(Please consult <<http://aws.amazon.com/whitepapers/>> for the latest version of this paper)

Contents

Contents	2
Abstract	3
Introduction	3
Development Phase	4
Source Code Repository	4
Project Management Tools	5
On-Demand Development Environments	6
Stopping vs. Terminating Amazon EC2 Instances	8
Integrating with AWS APIs and IDE Enhancements	8
Build Phase	8
Nightly Builds	9
On-Demand Builds	9
Storing and Distributing Build Output	10
Testing Phase	10
Automating Test Environments	11
Provisioning Instances	11
Provisioning Databases	12
Provisioning Complete Environments	12
Load Testing	12
Network Load Testing	13
Load Testing for AWS	13
Cost Optimization with Spot Instances	14
User Acceptance Testing	14
Side-by-Side Testing	15
Fault-Tolerance Testing	15
Resource Management	16
Cost Allocation and Multiple AWS Accounts	16
Conclusion	17
Further Reading	17

Abstract

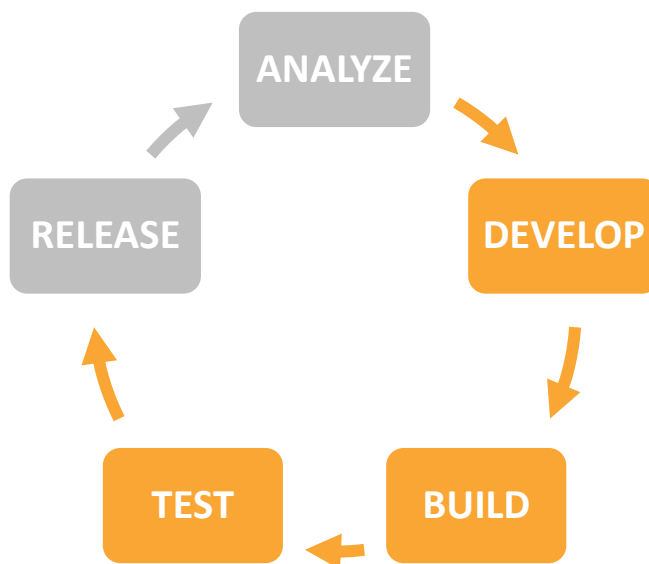
This whitepaper describes how Amazon Web Services (AWS) adds value in the various phases of the software development cycle, with specific focus on development and test. For the development phase, it shows how to use AWS for managing version control; it describes project management tools, the build process, and environments hosted on AWS; and it illustrates best practices. For the test phase, it describes how to manage test environments and run various kinds of tests, including load testing, acceptance testing, fault tolerance testing, etc. AWS provides unique advantages in each of these scenarios and phases, allowing you to pick and 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.

Introduction

Organizations write software for various reasons, ranging from core business needs—the organization is a software vendor—to customizing or integrating software. Organizations also create different types of software: web applications, standalone applications, automated agents, etc. In all such cases, development teams are pushed to deliver software of high quality and 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 life cycle typically consists of the following main elements:



This whitepaper covers aspects of the development, build, and test phases. For each of these phases, different types of IT infrastructure are needed. This is where 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 3–5 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 onto resources. This time could be spent focusing on the main job at hand.

By provisioning only the resources needed for the duration of development phases or test runs¹, your company can achieve important savings compared to investing upfront 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, in the development phase, we discuss how to securely and durably set up tools and processes such as version control, collaboration environments, and automated build processes; in the testing phase, we discuss how to set up test environments in an automated fashion, and how to run various types of test 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, etc., but also facilitates the execution 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 [Amazon Elastic Compute Cloud²](#) (Amazon EC2) instance and remotely installing version control software on it. Within minutes, developers can create Amazon EC2 instances, which are virtual machines over which they 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. Once you've properly installed and configured the source code repository instance, we recommend you create an

¹ Or complete test campaigns

² See: <http://aws.amazon.com/ec2/>

AMI from this setup in order to be able to quickly recreate that instance without having to reinstall and reconfigure the version control software.

You can store the repository's data separately from the host system to simplify maintenance or migration operations. [Amazon Elastic Block Store](#)³ (Amazon EBS) provides off-instance storage volumes that persist independently from the life of an instance. Once 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](#)⁴ ([Amazon S3](#)), a highly durable and scalable data store. Objects in Amazon S3 are redundantly stored on multiple devices across multiple facilities in an Amazon S3 Region. You can then use these snapshots as the starting point for new Amazon EBS volumes, and can protect your data for long term durability. By using EBS snapshots, you can easily manage backups of the source code repository. In case of a failure, you can recreate the repository data volume from the snapshots durably stored in Amazon S3, and recreate the source repository instance from an AMI.

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

As the source code repository grows and requires more storage capacity, two solutions are available: provision and attach additional Amazon EBS volumes to the repository instance, or provision a new larger Amazon EBS volume (up to 1 TB) based on a recent snapshot of the existing repository data. The new volume will then replace the existing volume, which you can delete. In both cases, those new Amazon EBS volumes are additional capacity that you create on-demand, in a matter of minutes.

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, etc. Most of the time, those tools are provided as web applications. Like any other classic web application, they require a server to run and also frequently a relational database. Both components can be installed on Amazon EC2 instances following similar deployment procedures to those used in classic environments, with the database using Amazon EBS volumes for data storage.

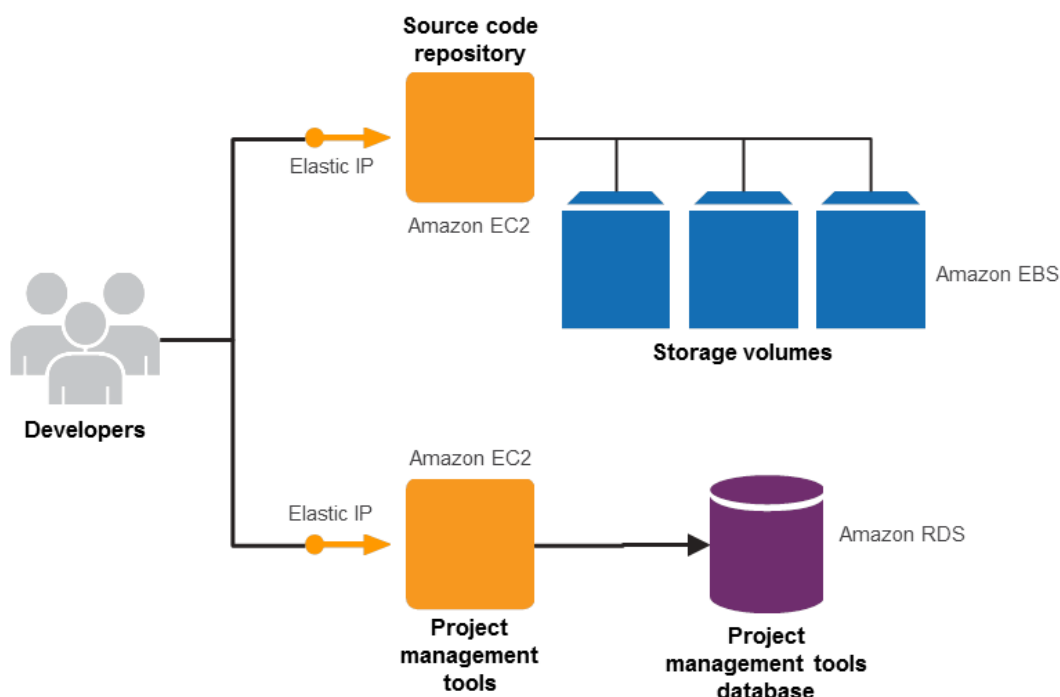
Project management tools have the same needs as source code repositories: they need to be available and data has to be durably stored. While you can mitigate the loss of code analysis reports by recreating them against the desired repository version, losing project or issue tracking information might 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, just as you would for source code repositories. Achieving proper durability for a database requires more effort and more attention: even by using Amazon EBS volumes, snapshots must be taken on a frozen file system in order to be consistent. Also, restoring a database might require additional operations other than restoring a volume from a snapshot and attaching it to an Amazon EC2 instance.

³ See: <http://aws.amazon.com/ebs/>

⁴ See: <http://aws.amazon.com/s3/>

To facilitate this part, 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 also make sure 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.

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 create a new web application server from the AMI on a more powerful Amazon EC2 instance type, and replace the previous server. Amazon RDS DB Instances can scale compute and memory resources with a few clicks on the AWS Management Console.



Note: For even quicker and easier deployment, many project management tools are available from the [AWS Marketplace](#)⁵ or as [Amazon Machine Images](#)⁶.

On-Demand Development Environments

Developers primarily use their local laptops or desktops to run their development environments. This is typically where the IDE is installed, where unit tests are run, where source code is checked in, etc. However, there are also a few cases where on-demand development environments hosted in AWS are helpful.

Some development projects may make use of specialized sets of tools where it would be cumbersome or resource intensive to install and maintain these on local machines, especially if such tools are used infrequently. For such cases, you can prepare and configure development environments with required tools (development tools, source

⁵ See: <https://aws.amazon.com/marketplace/b/2649274011/>

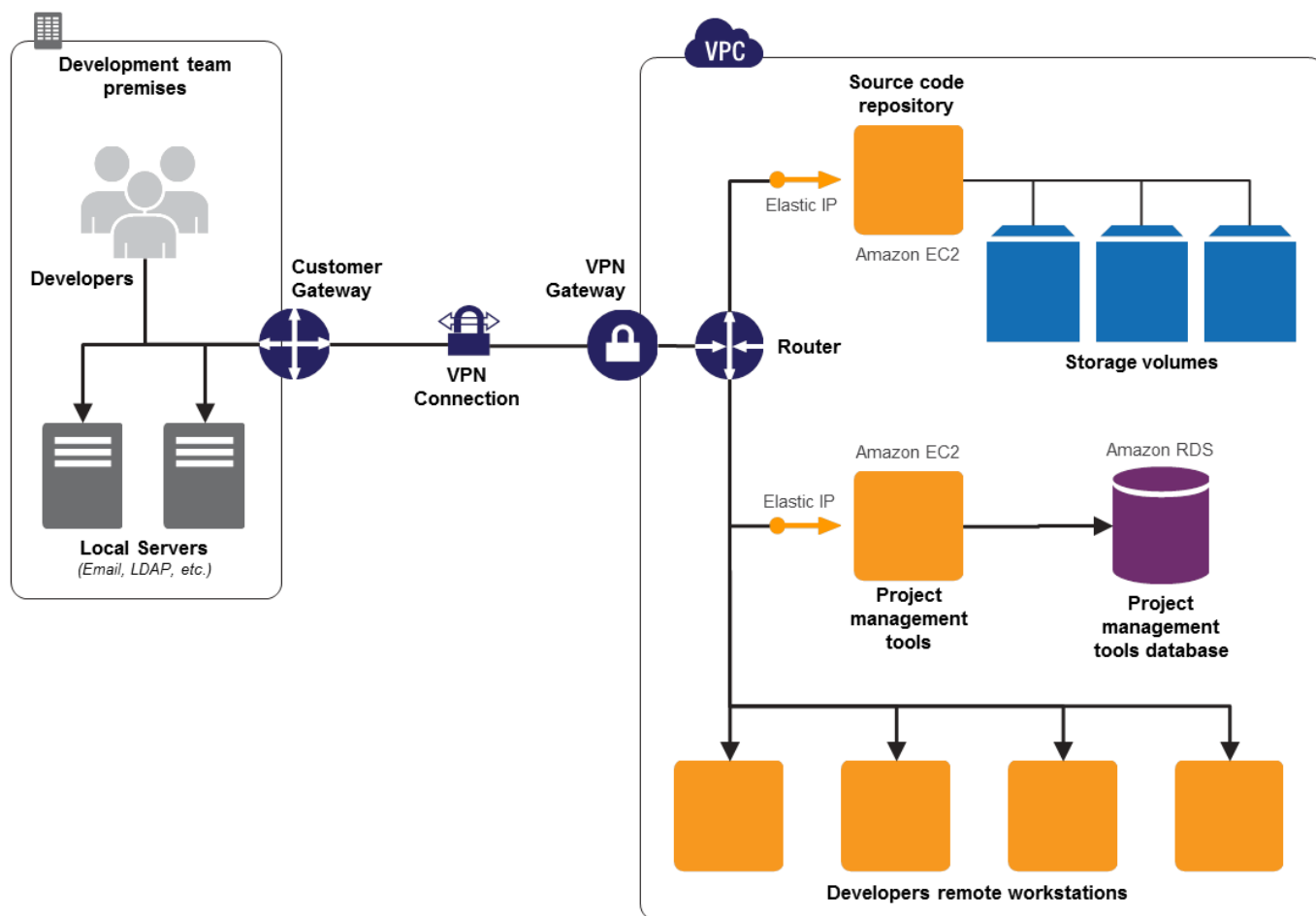
⁶ See: <https://aws.amazon.com/amis>

control, unit test suites, IDEs, etc.), and then bundle them as AMI(s). When an environment is needed, you can easily start the right environment and have it up and running in minimal time and with minimal effort. Then, 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 in production as well.

The concept of hosted desktops is not limited to development environments, but 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 below, in the context of setting up test environments. In many cases, such environments are set up within [Amazon Virtual Private Cloud](#) (Amazon VPC), which lets you extend your on-premise 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-premise resources (such as 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.



Stopping vs. Terminating 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: stop the instances, which is roughly equivalent to hibernating the operating system; or terminate them, 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—this is equivalent to reopening the lid of a laptop. (Note: any data stored on ephemeral drives will not be available after a stop/start sequence).

When you terminate 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. A terminated instance doesn't exist anymore and needs to be recreated from an AMI if needed. You would typically terminate the instance of a development environment if all work has been committed and/or the specific environment will not be used anymore.

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, thereby taking the complexity out of coding directly against a web service interface and dealing with details around authentication, retries, error handling, etc. The AWS SDK Tools are available for multiple languages: [Java](#)⁷, [.Net](#)⁸, [PHP](#)⁹, [Ruby](#)¹⁰, and for [mobile platforms](#)¹¹ Android and iOS.

AWS also offers [tools](#)¹² like the AWS Toolkit for Visual Studio and the AWS Toolkit for Eclipse, making it easier for you to interact with AWS from within your IDEs.

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, etc. Some projects might even require building the deliverables for multiple CPU architectures, platforms, or operating systems. At the end, 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 integration¹³ where every commit to the source repository triggers an automated build, followed by test suites.

⁷ See: <http://aws.amazon.com/sdkforjava/>

⁸ See: <http://aws.amazon.com/sdkfor.net/>

⁹ See: <http://aws.amazon.com/sdkforphp/>

¹⁰ See: <http://aws.amazon.com/sdkforruby/>

¹¹ See: <http://aws.amazon.com/mobile/>

¹² See: <http://aws.amazon.com/developertools/>

¹³ See: http://en.wikipedia.org/wiki/Continuous_integration

Nightly Builds

To mitigate this problem, teams working on projects with lengthy build times often adopt the “nightly build” (or neutral build¹⁴) approach, or break the project into smaller sub-projects (or even 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, as already explained in the previous sections, thus reducing build time by providing more CPU or memory capacity when it is 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 Amazon EC2 to use multiple build instances (worker nodes). Every time a new build is requested by the development team or triggered by a new commit to the source code repository, the build process is distributed to the 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 would pick the next build to process whenever they are free. To implement this system, [Amazon Simple Queue Service](#)¹⁵ (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 would commit code to the source code repository, which in turn pushes a build message into an Amazon SQS queue. This queue is polled by worker nodes. A worker node will 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](#)¹⁶ 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](#)¹⁷, 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 scenario¹⁸:

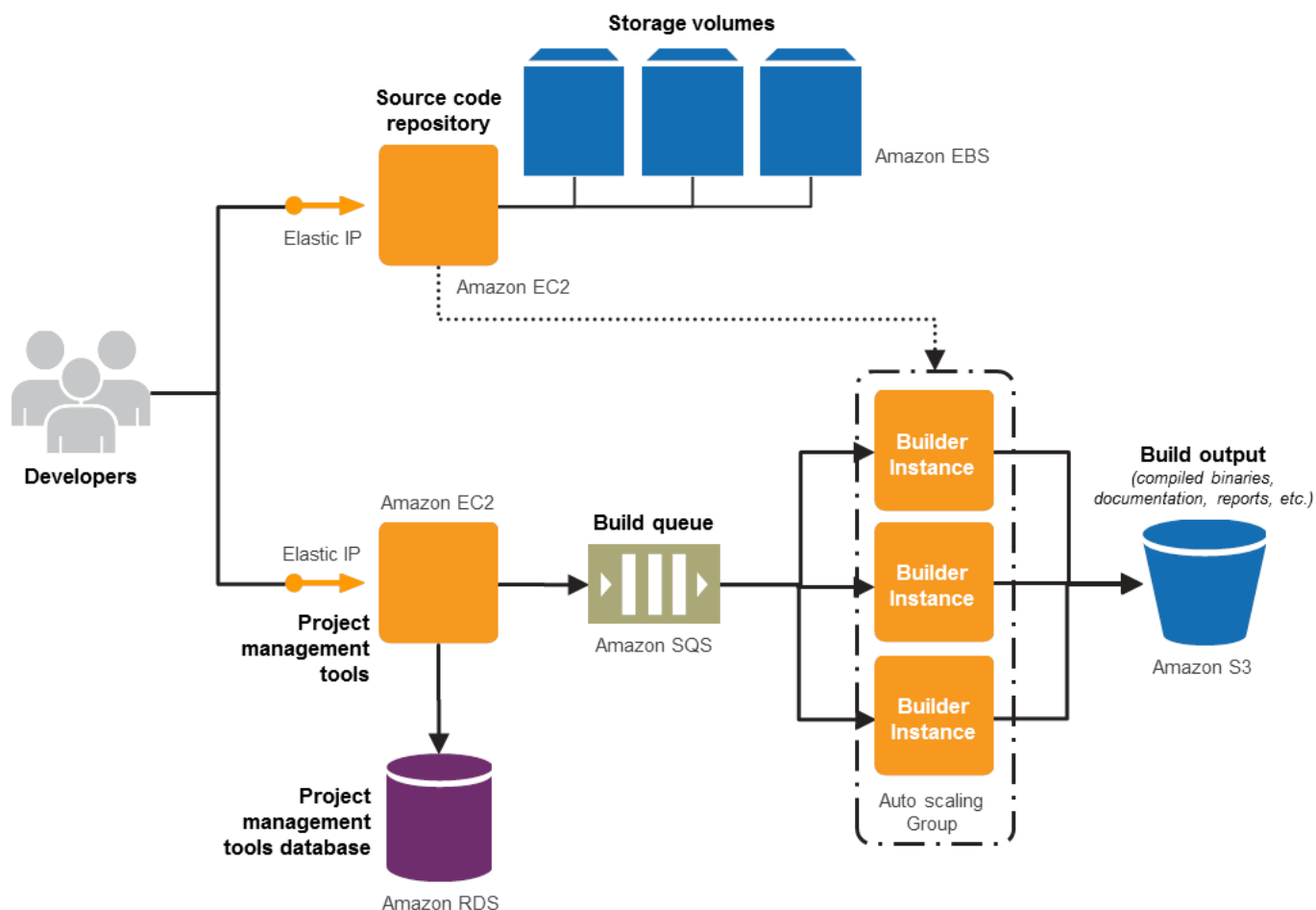
¹⁴ See: http://en.wikipedia.org/wiki/Nightly_build

¹⁵ See: <http://aws.amazon.com/sqs/>

¹⁶ See: <http://aws.amazon.com/autoscaling/>

¹⁷ See: <http://aws.amazon.com/cloudwatch/>

¹⁸ A more generic setup is described in the Reference Architecture “Batch processing”. See: http://d36cz9buwru1tt.cloudfront.net/architecturecenter/AWS_ac_ra_batch_03.pdf



Storing and Distributing Build Output

Every time you produce a build, you need to store the output somewhere. For this, Amazon S3 is an appropriate service. 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 Amazon S3 are particularly attractive. When you no longer need the build output, you can delete it, or use Amazon S3's data retention policies to do it.

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 Amazon S3, either publicly, or by configuring bucket policies and/or ACLs to restrict the distribution. Another option is to use [Amazon CloudFront](#)¹⁹, 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 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

¹⁹ See: <http://aws.amazon.com/cloudfront/>

forms: unit tests, performance tests, user acceptance tests, integration tests, etc., and all require IT resources to run. Thus, test teams face the same challenges as development teams: the need to have enough IT resources, but only during the limited duration of the test runs. Besides, test environments change frequently and are different from project to project, and as a result may well require different IT infrastructure or have varying capacity needs.

AWS's 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 need to be run.

Automating Test Environments

There are many software tools and frameworks available for automating the process of running tests, but in order to run those tests, proper infrastructure has to be in place. This involves provisioning infrastructure resources and initializing them with a sample dataset, deploying the software to be tested, orchestrating the test runs, and collecting results. The challenge here is not only to have enough resources to deploy the complete application with all the different servers or services it might require, but also 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 operate AWS programmatically using the AWS APIs or the command line interface (CLI) tools. Tasks that require human intervention in classic environments (allocating a new server, allocating and attaching storage, allocating a database, etc.) can be fully automated on AWS. 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 principles²⁰. Every time a successful build is produced, a test environment can be provisioned and automated tests executed 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 AMIs²¹. An AMI encapsulates the operating system and any other software or configuration files, pre-installed on the instance. When you launch the instance, all of the applications are already loaded from the AMI and ready to run. For information about creating AMIs, refer to the [Amazon EC2 documentation](#)²². 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, this quickly imposes having to define a strategy for managing and maintaining multiple versions of AMIs.

²⁰ See: http://en.wikipedia.org/wiki/Continuous_integration

²¹ See: <http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/launching-an-instance.html>

²² See: <http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/creating-an-ami.html>

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, etc.). More volatile components, like the application under development, are fetched and deployed to the instance at run-time. For more details on how to create self-bootstrapped instances, see the document "[Bootstrapping Applications on AWS](#)"²³

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 instance²⁴. From then on, every time you need a test database for a test environment, you can easily create one as an Amazon RDS instance from that initial snapshot²⁵. Each Amazon RDS instance started from the same snapshot will contain the same dataset, which helps ensure that your tests will be consistent.

Provisioning Complete Environments

While you can create complex test environments containing multiple instances by using the AWS APIs, command line tools, or the AWS Management Console, [AWS CloudFormation](#)²⁶ 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 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 allowing 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, etc.

For more information about how to create and automate deployments on AWS using AWS CloudFormation, visit: <http://aws.amazon.com/cloudformation/aws-cloudformation-articles-and-tutorials/>

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 web sites are specifically created to provide a service for a limited time: ticket sales for sports events, special sales (Black Friday), limited edition launches, etc. Such web sites need to 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 (example: search results must be returned in under 100 ms for up to 10,000 concurrent requests), and load tests are exercised in order to make sure that the system is able to 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

²³ See: <https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf>

²⁴ See: http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html

²⁵ See: http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_RestoreFromSnapshot.html

²⁶ See: <http://aws.amazon.com/cloudformation/>

performed in order to measure the behavior under heavy load conditions. The objective is to gradually increase the load of a system in order 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 amount of concurrent incoming requests, a huge dataset to process, etc. One of the main difficulties in load testing is being able to generate large enough amounts of inputs to push the tested system to its limits. Typically, it implies having large amounts of IT resources to deploy the system to test, and to generate the test input, which requires further infrastructure. Since 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, in order to make sure that each major change to the project doesn't adversely affect system performance and efficiency. Conversely, by launching automated load tests, it's easy to check if 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: <https://aws.amazon.com/marketplace/b/2649283011/>

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 well 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. It's important to keep in mind the characteristics of different instance types: as a general rule, larger instance types provide more 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. Since you can terminate load-testing Amazon EC2 instances right after the test, you should transfer log data to Amazon S3 for storage and later analysis.

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 Balancing²⁷ configurations are correct. Auto Scaling offers many parameters²⁸ 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-

²⁷ Read the technical article "Best Practices in Evaluating Elastic Load Balancing" available at <http://aws.amazon.com/articles/1636185810492479>

²⁸ Refer to the Auto Scaling documentation: <http://docs.amazonwebservices.com/AutoScaling/latest/DeveloperGuide/Welcome.html>

provisioning time might also affect an application's ability to scale up quickly enough. After the initialization of the operating system running on Amazon EC2 instances, additional services are initialized—such as web servers, application servers, memory caches, middleware services, etc.. 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 will provide 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 like consumers polling an Amazon SQS queue²⁹ or workers and deciders participating in an Amazon Simple Workflow Service (Amazon SWF) workflow³⁰. In both cases, load testing the system can help you ensure you've correctly implemented and configured Auto Scaling groups or other automated scaling techniques, in order 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](#)³¹ 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, you place a Spot Instance request, specifying the instance type, the desired Availability Zone³², the number of Spot Instances to run, and the maximum price to pay per instance hour. The Spot Price history for the past 90 days is available via the Amazon EC2 API and the AWS Management Console³³. If the maximum price bid exceeds the current Spot Price, the request is fulfilled and instances will be started and run until either they are terminated or the Spot Price increases above the maximum price (whichever is sooner).

For more information about Spot Instances, visit: <http://aws.amazon.com/ec2/spot-instances/>. A case study on using Spot Instances for load-testing is available here: <http://aws.amazon.com/solutions/case-studies/browsermob/>.

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 grey 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, since 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

²⁹ See the blog article: <http://aws.typepad.com/aws/2011/07/additional-cloudwatch-metrics-for-amazon-sqs-and-amazon-sns.html>

³⁰ See the blog article: <http://aws.typepad.com/aws/amazon-simple-workflow-service/>

³¹ See: <http://aws.amazon.com/ec2/spot-instances/>

³² An Availability Zone is a distinct location within a Region that is engineered to be insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

³³ Current spot price is available here: <http://aws.amazon.com/ec2/spot-instances/#6>

needed in a cost-effective way, which is also very appropriate for acceptance testing. Using some of the techniques described above, 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. Besides, such an application can also be integrated with other production services inside the company, like LDAP, email servers, etc., 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 if 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 make sure your changes positively impact the global performance of the system.

Once 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 they are needed, taking advantage of the pay-as-you-go model of AWS. In contrast, with traditional hardware, it may not easily 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/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, etc. 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 Zone³⁴ option that enhances database availability by automatically provisioning and managing a replica in a different Availability Zone. For more information on how to

³⁴ See: <http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/Concepts.DBInstance.html#Concepts.MultiAZ>

build fault-tolerant architectures running on AWS, read the whitepaper “[Building Fault -Tolerant Applications on AWS](#)”³⁵ and the resources available in the [AWS Architecture Center](#)³⁶.

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 command line interface to interact with the test environment. For example, you might terminate Amazon EC2 instances, and thereby test if an Auto Scaling group is working as expected and a replacement instance automatically provisioned. You can also automate this kind of test. It is a best practice is to use automated tools that, for example, occasionally and randomly terminate Amazon EC2 instances.

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 instances can be done easily using AWS CloudFormation stacks or some of the other automation techniques described. 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 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 (e.g., Amazon VPC), security configurations (e.g., Security Groups and IAM credentials)
- Monitoring resource usage and charges using Amazon CloudWatch, and allocating these to team budgets

While you can use the AWS Management Console to achieve the tasks above, 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, for example, 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, for example, to

³⁵ Available at: http://d36cz9buwru1tt.cloudfront.net/AWS_Building_Fault_Tolerant_Applications.pdf

³⁶ Visit: <http://aws.amazon.com/architecture/>

simplify payments. Another option is to make use of the 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 used³⁷.

Amazon Web Services 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 needed, instantly scale up as the workload grows, and release resources when they are no longer needed. Whether development and test environments consist of a couple of 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 allows you to focus on your project, not on operating and maintaining your infrastructure.

AWS also enables possibilities that were 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 up-front expenses or long-term commitments, making AWS a platform of choice for development and test.

Further Reading

- Whitepaper: “Operational Checklists for AWS”
http://media.amazonwebservices.com/AWS_Operational_Checklists.pdf
- Whitepaper: “How AWS Pricing Works”
http://media.amazonwebservices.com/AWS_Pricing_Overview.pdf
- AWS Architecture Center – <http://aws.amazon.com/architecture>
- AWS Technical Whitepapers – <http://aws.amazon.com/whitepapers/>

³⁷ See: <http://aws.amazon.com/economics/>