AWS AMI DESIGN

Overview

Amazon Web Services (AWS) offers its customers several methods that make it easier to provision Amazon Elastic Compute Cloud (Amazon EC2) instances and store instance configurations across a variety of different server and application deployment models. The most common unit of management is the Amazon Machine Image (AMI), which provides the information required to launch an EC2 instance. AWS customers specify an AMI when launching an instance and can use a single AMI to launch multiple EC2 instances. This document provides AWS customers with strategic guidance and common methods for determining what software to include in the image when building custom AMIs.

The following sections assume basic knowledge of Amazon EC2 and supported operating systems, image-based OS deployment, and configuration management systems.

General Best Practices

All organizations should have a documented process for provisioning server images to ensure images can be recreated and easily updated in adherence to corporate standards. When determining what to include in an AMI, consider the following AMI design best practices:

- Avoid embedding passwords, private keys, or other sensitive information in AMIs.
- Leverage AWS CloudFormation or a third-party configuration management tool to document and automate AMI creation and updating.
- Create a library of reusable, modular templates that can be programatically assembled to create different types of AMIs.
- Instrument AMIs with a standard bootstrapping capability that allows the instance to reference runtime information at launch.
- Develop a consistent strategy for tagging AMIs to allow for the easy organization and identification of images and their contents.

AMI Design Options

AMI design options fall along a spectrum of deployment simplicity in relation to deployment flexibility. The simplest AMIs are fully baked and purpose-built to deploy a complete running instance, including the installation and configuration of all required software. However, this approach limits flexibility, as a fully baked AMI can only be used to deploy a single instance or a farm of identical instances. The most flexible AMIs include only minimal configurations and software before then dynamically installing the required packages on first boot. This approach trades simplicity for flexibility as each instance must be properly bootstrapped before it can function as intended.

The ideal AMI design depends largely on the constraints of the workload. When considering which option is right for your organization, keep the following questions in mind:

- How quickly do you need to be able to recover a failing instance or add additional compute capacity?
- Does the workload’s baseline stay static for a relatively long period?
- Does your server configuration require manual provisioning or configuration?
- Do you need to minimize the complexity of deploying resources to both AWS and on-premises environments?
- Are there existing server provisioning tools or processes that you are trying to align with AWS?
Fully Baked AMI
On one end of the spectrum is the most common method for AMI design that provisions a fully functional instance, including all necessary software for a specific role or even a single instance. Organizations often create a baseline AMI that conforms to minimal security and configuration requirements, and then build on this baseline to create fully baked AMIs that are specific to applications or infrastructures for actual instance deployment.

Fully baked AMIs are the simplest to deploy and provide the fastest launch times. For this reason, fully baked AMIs are useful to quickly deploy replacement instances or add additional capacity. Fully baked AMIs also require minimal changes when launched and can be used to capture manual installation and configuration steps that cannot be automated. The process of using fully baked AMIs is similar to how many organizations deploy virtual servers in their existing data centers, which makes this simple approach applicable to customers who are new to the AWS platform and are accustomed to image-based server deployments.

Considerations
At scale, it can be expensive and cumbersome to maintain unique AMIs for every instance or group of instances. This approach is therefore more suitable for smaller AWS deployments or when combined with an automated AMI build and management system. Even though this approach deploys a fully functional system, AWS highly recommends including a mechanism to read and process EC2-instance user data at launch. It is often advantageous to be able to pass launch instructions for minor configuration changes to your OS or application, such as those specific to a region, availability zone, or lifecycle environment.

JeOS AMI
At the other end of the spectrum is the just enough operating system (JeOS) approach. This approach creates an AMI that combines a minimal operating system with a configuration management agent that builds a fully functional system at instance launch. On first boot, the configuration agent downloads, installs, configures, and integrates all required software.

JeOS AMIs offer the most flexibility during deployment and the highest levels of portability because they leverage minimal server images. Customers who prefer this approach typically have experience with AWS and configuration automation tools. They want deployment flexibility to be able to pull the latest software builds and updates at launch, or they might require minimal complexity in order to deploy resources to both AWS and on-premises environments.

Considerations
Downloading, installing, and configuring all required software on first boot can significantly increase instance launch times, making this approach less suitable for quickly replacing a failed instance or adding capacity. Also, JeOS AMIs require tight coupling with a configuration management system and can be more difficult to use with legacy software that typically has limited automation capabilities.
Hybrid AMI
Hybrid AMIs provide a subset of the software needed to produce a fully functional instance, falling in between the fully baked and JeOS options on the AMI design spectrum. This approach creates a partially baked, generic infrastructure AMI that is further configured on first boot based on specific application requirements. The decision about what to bake into the AMI and what to install and configure at launch is typically based on how reusable the underlying software package is across all instances and how long it takes to download, install, and configure the software component. For example, to reduce instance launch times, include server management agents and baseline security configurations into an AMI. To further reduce instance launch times, include the necessary software to create role-specific AMIs (e.g., web servers, application servers, database servers).

Hybrid AMIs combine the flexibility of post-launch changes with the speed of preinstalled and preconfigured infrastructure components. This can be especially useful for merging frequently changing software (such as application code or security updates) with infrequently changing software (such as management agents and security baselines). Customers who use this approach typically have a moderate amount of experience with AWS and configuration automation tools, and want to combine the deployment simplicity of a baked image with the deployment flexibility that pulls the latest software builds and updates at launch.

Considerations
Be aware that each change to software that is tightly coupled with the AMI requires a new AMI to be built. Although this approach requires fewer AMIs to maintain than the fully baked AMI approach, it will still result in more AMIs to maintain than the JeOS approach. We therefore recommend that customers who take this approach combine it with an automated AMI build-management system and a configuration management tool.

Example AMI Tagging Structure
In addition to choosing the appropriate AMI design pattern for your organization, it is important to have a consistent strategy for tagging AMIs to allow for the easy organization and identification of AMIs and their contents. The following are suggested AMI tags for technical, business, and security purposes.

**Technical Tags**
- **Name** – A simple identifier for individual AMIs
- **Version** – An identifier to help distinguish between AMIs with similar names
- **Software** – A list of software included in the AMI
- **Role** – The role of an instance launched from this AMI (e.g., web server, message broker)
- **System** – The IT or application system into which this AMI will be included (This is especially useful for fully baked AMIs that embed system-specific configuration information.)

**Business Tags**
- **Project** – The specific project(s) for which this AMI was created
- **Cost Center/Business Unit** – The cost center or business unit associated with the AMI
- **Stage** – The development stage of the AMI to help identify if it is appropriate for production use

**Security Tags**
- **Confidentiality** – An identifier for the specific data-confidentiality level of the AMI (Useful for organizations that want to embed additional security controls in instances that process particular levels of classified data.)
- **Compliance** – An identifier for AMIs designed to adhere to specific compliance requirements
## Resources

<table>
<thead>
<tr>
<th>Documentation Type</th>
<th>Documentation URL</th>
<th>Description</th>
</tr>
</thead>
</table>