HELECLOUD

Building secured immutable infrastructure on AWS

Daniel Rankov
2018

HELECLOUD

Table of Contents

Building secured immutable infrastructure on AWS 1
Abstract 3
Purpose 4
Infrastructure as Code 5
Creating auditable AWS architecture 5
Immutable infrastructure 7
Building images process 10
Tools to achieve AMI Factory 11
Implementation of AMI Factory 12
Scheduled scan 13
Configuration management 14
Instance bootstrap 15
Testing and fast feedback 15
Conclusion 17
References 18
Further readings 18

Abstract

This whitepaper is intended for solutions architects and developers who are building solutions that will be deployed on Amazon Web Services (AWS). It provides an overview of establishing the foundation of immutable infrastructure by creating predefined images and performing regular security scans. This whitepaper is not intended as a step-by-step setup guide.

HELECLOUD

Purpose

Purpose of the whitepaper is to present a way to achieve a fully auditable and secure deployment in AWS by using Infrastructure as Code and Immutable infrastructure approach.

The solution is based on creating continuous secure pipeline of software and infrastructure delivery.

The paper is focused on achieving immutable infrastructure processes and tools. As this is repeatable process a term AMI Factory is introduced. AMI Factory represents the implementation of building AMIs to achieve immutable infrastructure and performing security scans.

The whole infrastructure is regularly scanned for vulnerabilities in cost effective way.

By following the infrastructure as code and immutable infrastructure approach one does not have to login neither to theirs AWS account, nor to the instances deployed. Consistent deploy, fast revert and fast scale can be achieved.

Following a strict infrastructure and configuration as code – everything is versioned, reviewed, verified, the right telemetry is being gathered, fast deployment and fast feedback loops are achieved. Security and audit are injected and shifted left in the delivery pipeline in DevSecOps manner.

"One approach that we learned quickly is that to build secure services, it is necessary to integrate security at the very beginning of service design. The security team is not a group that does validation after something has been built. They must be partners on day one to make sure that security is fundamentally rock solid from the ground up." [1] Werner Vogels

Infrastructure as Code

Every service in AWS can be managed by API call. By using the API one can build reliable, predictable and fully automated service management at scale. The approach is known as Infrastructure as Code (IaC) – programmable infrastructure - helping you create, manage and configure infrastructure. The process of developing infrastructure follows standard development processes like version control, testing, deployment, etc. Peer reviews may be part of the release process, as well as security checks. Tools to achieve IaC are CloudFormation, Terraform on declarative front, or Ansible, Chef, SaltStack, and any of the AWS SDK on the procedural front, one may choose[2], the best fit for the organization.

HashiCorp Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment. CloudFormation allows you to use a simple text file to model and provision, in an automated and secure manner, all the resources needed for your applications across all regions and accounts. This file serves as the single source of truth for your cloud environment.

Creating auditable AWS architecture

Following Infrastructure as Code approach one can create a fully auditable, repeatable and consistent AWS infrastructure.

It's auditable – everyone can check the code.

It's repeatable – same code can be executed multiple times over different environments and AWS accounts.

It's consistent - executing the code guarantees the same result over and over again, even if somebody does a manual change – it will be overwritten.

Using repeatability feature makes it easy to isolate different environments. In a standard approach there would typically be a Development, UAT, Production environments, and they can be all the same by reusing the same code.

Infrastructure changes are introduced by code, so a pipeline may be built. Having this pipeline to be the central place to execute changes means that every other access can be limited. The users with AWS account access may be limited to read-only roles.

The different environments may be isolated on VPC level, even on subnet, but even better approach would be to isolate the workloads in dedicated AWS accounts. Doing that guarantees that not any resources are shared. Additionally, AWS limits will be specific per account, this way if too many resources are used in Development account this will not affect the Production in any way.

These are classic references when designing AWS workloads: AWS Well-Architected Framework and to AWS Security Best Practices with Shared Responsibility Model.

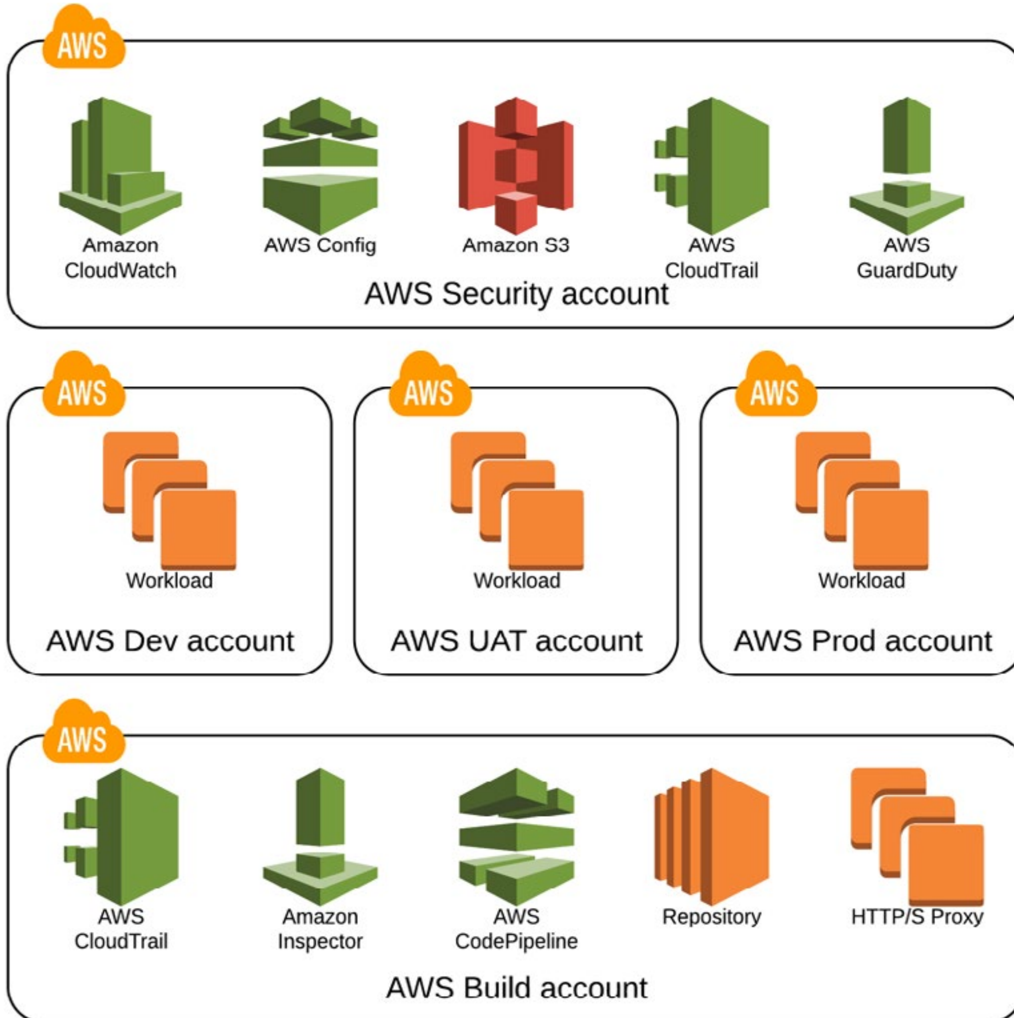
Part of hardening an AWS account is to achieve a CIS compliance.

https://aws.amazon.com/quickstart/architecture/compliance-cis-benchmark/

https://d0.awsstatic.com/whitepapers/compliance/AWS_CIS_Foundations_Benchmark.pdf

HELECLOUD

Let's use this AWS account structure as example for deploying workload, review on every account follows.



AWS Security account

A dedicated AWS account may be created for security and audit purposes. Access to this account is limited to Security team only, it might be as well a read-only access. This account contains all the log files from all other AWS accounts that your organization handles including Amazon CloudWatch Logs, AWS CloudTrail, and VPC Flow Logs.

One can have CloudTrail deliver log files from multiple AWS accounts into a single Amazon S3 bucket: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-receive-logs-from-multiple-accounts.html

VPC flow logs can be exported to Amazon S3 and Amazon Athena can be used to analyse the data: https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/flow-logs.html https://docs.aws.amazon.com/athena/latest/ug/vpc-flow-logs.html

AWS Config supports Multi-Account Multi-Region Data Aggregation: https://docs.aws.amazon.com/config/latest/developerguide/aggregate-data.html

GuardDuty is a managed threat detection service and may be configured to monitor multiple accounts: https://aws.amazon.com/blogs/security/how-to-manage-amazon-guardduty-security-findings-across-multiple-accounts/

HELECLOUD

AWS Application accounts – Development, UAT, Production

Every piece of infrastructure and services deployed in the AWS account is managed by code, making it easy reproducible, so one can create as many AWS accounts as needed in order to achieve the proper isolation of deployment.

Using separate AWS accounts per workload brings multiple advantages – easier IAM configuration, different AWS limits and more.

The workloads deployed in these AWS accounts may be in private only subnets with limited Internet access, where outgoing traffic, if needed can be fully blocked or filtered. In case outgoing traffic is needed a VPC peering to the AWS Build account can be created.

VPC Endpoints can be used to internally access AWS services without the need of Internet access. Some of the AWS services currently reachable are Amazon S3, DynamoDB, Kinesis, SNS, EC2 API, SSM, Secrets manager, CloudWatch Logs.

AWS Build account

This is the only AWS account that have outgoing Internet, used to download the proper content needed to build the software.

Access to Internet can be additionally filtered by using a Layer 7 HTTP/S filtering proxy like Squid.

A repository with artefacts is used, where all the needed software is downloaded and checked. The repository may also contain continuously updated Linux repository. Having the needed software in a central place makes it auditable, faster to retrieve, and available even if the remote site fails. Such repository might be Sonatype Nexus or JFrog Artifactory, they might be combined with additional security and audit scans.

This dedicated AWS account is used to create the AMI Factory and perform security scans.

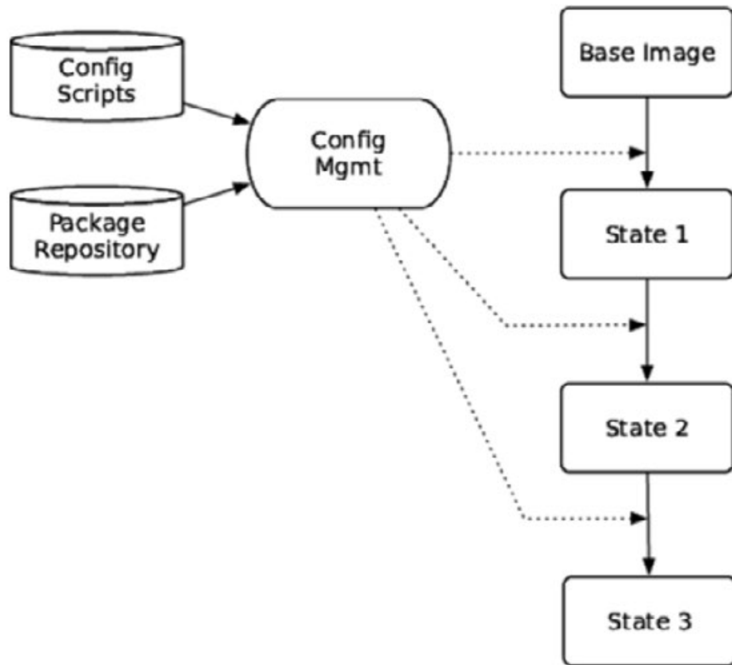
Immutable infrastructure

"In a traditional mutable server infrastructure, servers are continually updated and modified in place. Engineers and administrators working with this kind of infrastructure can SSH into their servers, upgrade or downgrade packages manually, tweak configuration files on a server-by-server basis and deploy new code directly onto existing servers. In other words, these servers are mutable; they can be changed after they're created. Infrastructure comprised of mutable servers can itself be called mutable, traditional, or (disparagingly) artisanal.

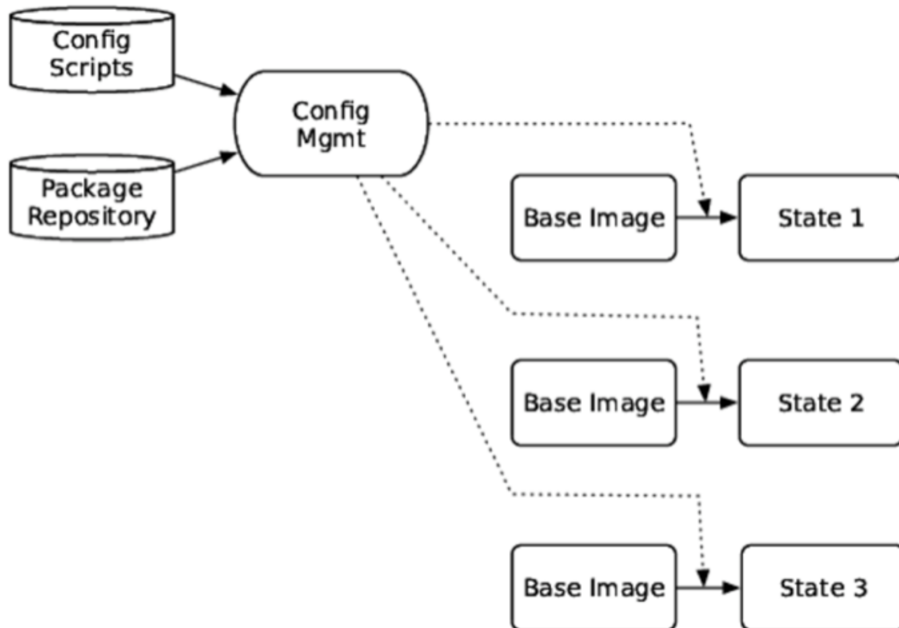
An immutable infrastructure is another infrastructure paradigm in which servers are never modified after they're deployed. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they're validated, they're put into use and the old ones are decommissioned."[3]

HELECLOUD

Traditional configuration management tools like Ansible, Puppet, Chef work like this:



The immutable infrastructure works by building images every time and the approach looks like this:



Immutable infrastructure has the following benefits:

- Consistency – images never change
- No snowflake servers and no configuration drift
- Faster server provisioning and horizontal scaling
- Predictable deployment process with simple rollback

Immutable infrastructure relies on working with images, where all the needed software is installed. The process of building images is known as baking. The process is repeatable and when automation is used the code is auditable and traceable. Image once build does not change – no new software nor binaries are installed – only environment specific configuration is applied once at instance startup (bootstrap in AWS is achieved by user_data).

Remote access (SSH/RDP) can be turned off or limited to "break glass" procedure only, so that nobody except security/forensics team can login to an instance.

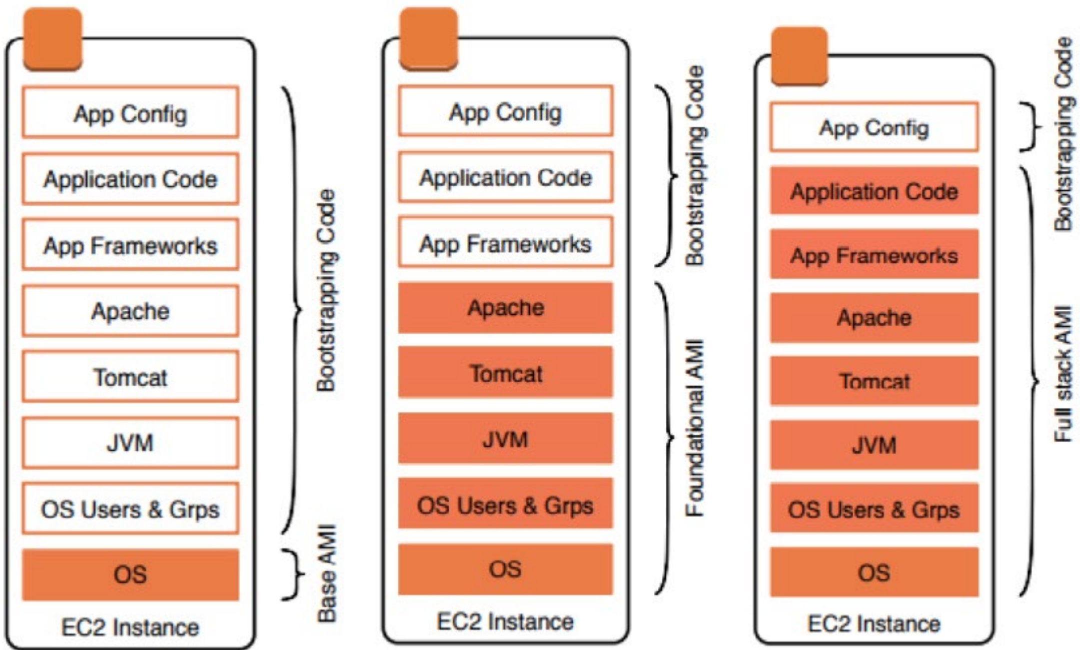
No patches are installed on live instances, no configuration is changed on live instances. This way there will be no configuration drift between servers.

Vulnerability scans may be performed on an image. While performing vulnerability scans on all instances is an option and sometimes a pricy one, when working with images only one instance of the image can be scanned to verify that there are no security issues. Performing security scan on a single instance instead of every image might have huge cost savings effect. Infrastructure is immutable, images don't change, so scanning a single image for CVEs is the same as scanning every image but is for a fraction of the cost.

HELECLOUD

Building images process

Following graphic illustrates the concept of creating static images. AWS whitepaper "Managing Your AWS Infrastructure at Scale" gives in depths on this process.



One always starts a base AMI, where only the OS is installed. In order to achieve immutable infrastructure, one should strive to achieve repeatable build process, where everything is installed.

Having a case where multiple AWS accounts are used for advanced isolation between Dev/UAT/Prod. An image can be built only once, then populated between the AWS accounts the only difference would be the configuration, nothing else! This creates an opportunity to make CVE scans on a single instance of an image, instead of scanning each and every instance deployed. This eliminates the overhead of running security scans on production instances and might have major cost benefits.

Tools to achieve AMI Factory

Packer is a product of HashiCorp used to automate the process of building AMIs. Packer follows a straight forward process of installing the needed software on an EC2 instance and registering it as AMI. Configuration is simply defined in JSON formatted structure.

Having AMI creation process as code gives the auditability and traceability of what has been (or not) installed.

AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. AWS CodePipeline automates the steps required to release your software changes continuously.

AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don't need to provision, manage, and scale your own build servers.

AWS Inspector has been chosen to perform regular security scans on Common Vulnerabilities and Exposures (CVEs). List of supported OS - https://docs.aws.amazon.com/inspector/latest/userguide/inspector_rule-packages_across_os.html

- The rules in Common Vulnerabilities and Exposures (CVE) package help verify whether the EC2 instances in your assessment targets are exposed to common vulnerabilities and exposures (CVEs). Attacks can exploit unpatched vulnerabilities to compromise the confidentiality, integrity, or availability of your service or data. The CVE system provides a reference method for publicly known information security vulnerabilities and exposures. For more information, go to https://cve.mitre.org/.

- Center for Internet Security (CIS) Benchmarks - The CIS Security Benchmarks program provides well-defined, un-biased and consensus-based industry best practices to help organizations assess and improve their security. Support for CentOS, RHEL, Ubuntu was recently announced - this is a full list of OS supported by AWS Inspector for CIS scans: https://docs.aws.amazon.com/inspector/latest/userguide/inspector_cis.html

- Security Best Practices - The rules in this package help determine whether your systems are configured securely.

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running.

Amazon S3 is object storage built to store and retrieve any amount of data from anywhere – web sites and mobile apps, corporate applications, and data from IoT sensors or devices. It is designed to deliver 99.999999999% durability, and stores data for millions of applications used by market leaders in every industry.

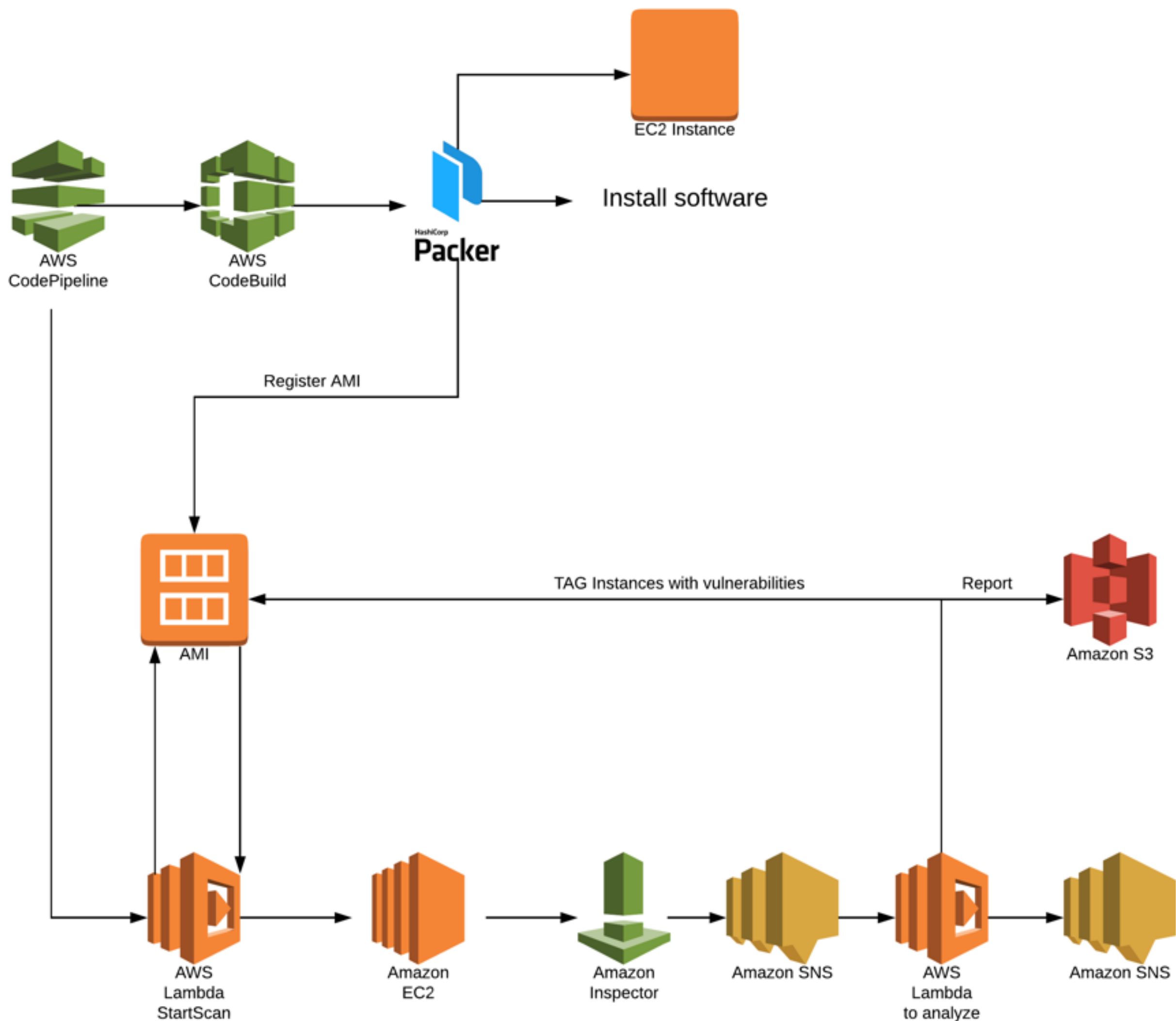
Amazon Simple Notification Service (SNS) is a flexible, fully managed pub/sub messaging and mobile notifications service for coordinating the delivery of messages to subscribing endpoints and clients.

HELECLOUD

Implementation of AMI Factory

In the following scenario a central AWS account is used as a Build account, where images (AMIs) are being build. These images may be used in the same AWS account in different VPCs or they can be populated to different AWS accounts and deployed there. The most beneficial is the image is always the same.

Building AMIs is a repeatable process and in order to be auditable it has to be fully automated, with no manual intervention.



The AMIFactory process is orchestrated by AWS CodePipeline.

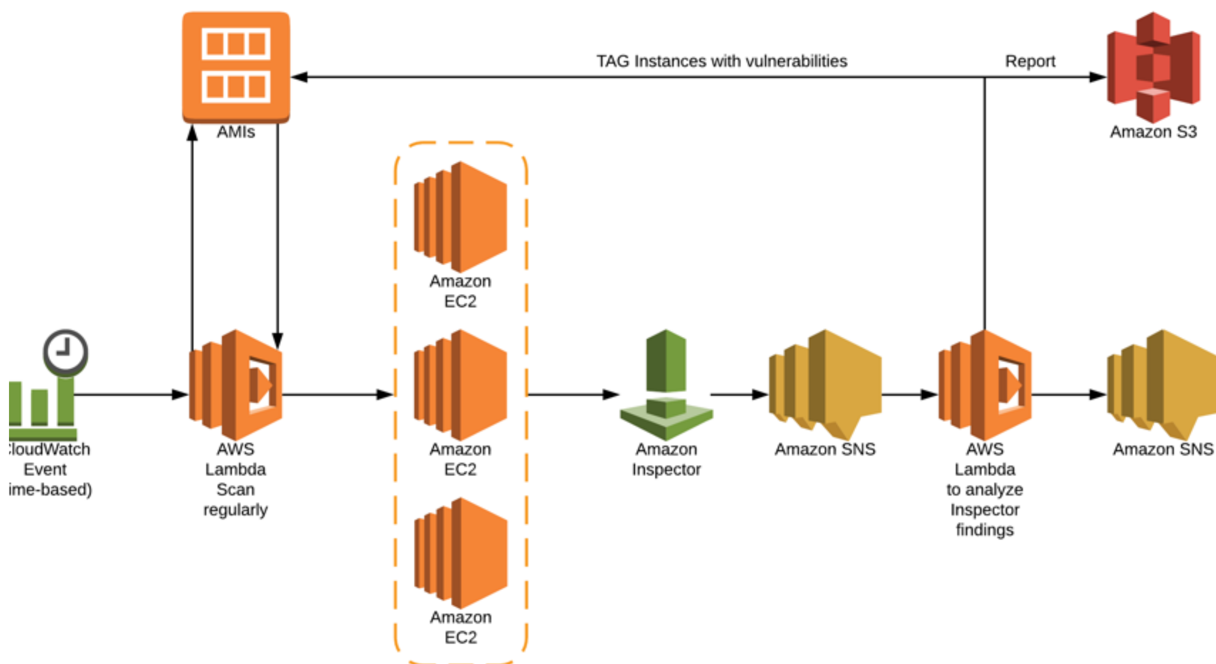
1) CodePipeline starts AWS CodeBuild pulling code from a repository it runs HashiCorp Packer
2) Packer process does:
   a. Start new EC2 instance
   b. Connect to that EC2 instance and executes predefined scripts
      i. OS update
      ii. Apply OS configuration and tuning
      iii. Apply CIS hardening on OS
      iv. Install application
      v. Install antivirus, IDS, IPS, file integrity check software
      vi. Install CloudWatch agent
      vii. Install Inspector agent

HELECLOUD

c. Register new AMI
3) AWS CodePipeline executes an AWS Lambda function to start EC2 instance from the newly created AMI. Tag is applied to it, so that on the next step AWS Inspector can scan only this instance.
4) Inspector scan for CVEs vulnerability scans is started on the EC2 instance
5) Inspector send notification to SNS when finished
6) AWS Lambda function which does the analyses of Inspector findings is triggered.
   a. Terminate the EC2 instance
   b. AWS Inspector Report is saved to S3 bucket
   c. If vulnerabilities are found for an AMI, a TAG is applied to it
   d. Findings are being sent to the Security team

Scheduled scan

Security scans are performed regularly to check AMIs for vulnerabilities. This scan should be run regularly – it might be done on daily or weekly bases.



CloudWatch Events rule is created to run every 7 days
1) AWS Lambda function scans your AWS account and finds AMIs in use
2) AWS Lambda starts EC2 instance for every found AMI in step 2 + the latest version of each AMI. Scanning the latest AMI builds confidence that it's ready for deploy.
3) EC2 instances are started
4) Inspector scans each of the instances for CVE
5) Inspector send notification to SNS when finished
6) AWS Lambda function which does the analyses of Inspector findings is triggered.
   a. AWS Inspector Report is saved to S3 bucket
   b. If vulnerabilities are found for an AMI, a TAG is applied to it
   c. Findings are sent to Security team

AMIs which are already tagged as vulnerable don't have to be scanned more than once.

Considerations when using scheduled scans:

HELECLOUD

- AWS account EC2 limits. Lambda function starting EC2 Instances is based on the count of AMIs. The more there are, the more instances are started, this might result in reaching EC2 AWS account limits. Limits can be changed, or the lambda function can be configured to start batches of instances. https://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2
- Cost. Starting multiple instances will result in additional cost. Good news is that AWS charges EC2 instances on per second basis. https://aws.amazon.com/about-aws/whats-new/2017/10/announcing-amazon-ec2-per-second-billing/

AMI lifecycle

Over time AMI count rises with every build, one should have a retirement strategy. In the usual case not, every AMI ever build should be kept. Building a working strategy per business case is the best approach. A Lambda function can do the periodic sanitization.

For compliance and audit reasons it might make sense to keep AMIs that has ever been deployed to Production environment for a prolonged period of time.

Configuration management

When working with immutable infrastructure and prebuild images where all the needed software is installed. The only difference between environments is the application configuration - it's best to have a central storage for configuration properties. Here are some options:

AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, and license codes as parameter values. You can store values as plain text or encrypted data. You can then reference values by using the unique name that you specified when you created the parameter. Highly scalable, available, and durable, Parameter Store is backed by the AWS Cloud. Parameter Store is offered at no additional charge.

Example of using Parameter Store is storing credentials like:

/testing/db_user

/testing/db_password

/production/db_user

/production/db_password

By using IAM privileges the access can be restricted:

https://aws.amazon.com/blogs/mt/the-right-way-to-store-secrets-using-parameter-store/

Both Terraform and CloudFormation support Parameter Store.

Amazon S3 bucket. Configuration files might be placed in encrypted S3 bucket and pulled at instance bootstrap. Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. Amazon S3 default encryption provides a way to

set the default encryption behaviour for an S3 bucket. You can set default encryption on a bucket so that all objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or AWS KMS-managed keys (SSE-KMS). Amazon S3 also supports access logging, versioning and cross region replication.

AWS Secrets Manager helps you protect secrets needed to access your applications, services, and IT resources. The service enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. Users and applications retrieve secrets with a call to Secrets Manager APIs, eliminating the need to hardcode sensitive information in plain text. Secrets Manager offers secret rotation with built-in integration for Amazon RDS for MySQL, PostgreSQL, and Amazon Aurora. Also, the service is extensible to other types of secrets, including API keys and OAuth tokens. In addition, Secrets Manager enables you to control access to secrets using fine-grained permissions and audit secret rotation centrally for resources in the AWS Cloud, third-party services, and on-premises. https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html

HashiCorp Vault secures, stores, and tightly controls access to tokens, passwords, certificates, API keys, and other secrets in modern computing. Vault handles leasing, key revocation, key rolling, and auditing. Through a unified API, users can access an encrypted Key/Value store and network encryption-as-a-service, or generate AWS IAM/STS credentials, SQL/NoSQL databases, X.509 certificates, SSH credentials, and more.

In case there are complicated configuration files that one like to build Ansible jinja templates may help. Ansible integrates with Parameter Store, where from variables can be pulled and substituted in the configuration templates.

Instance bootstrap

Launching an instance in Amazon EC2, there is the option of passing user data script to the instance that can be used to perform common automated configuration tasks and even run scripts after the instance starts. You can pass two types of user data to Amazon EC2: shell scripts and cloud-init directives. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html

By using user data scripts one can pull and apply the proper configuration applicable to the environment.

By assigning proper IAM Instance profile the access to AWS resources (Parameter Store, S3) may be fine-grain limited.

Testing and fast feedback

As part of the CodePipeline process a deployment step may be added, this way every new AMI is deployed in Dev/Test or Integration environment. With focus on fast feedback this step may be executed simultaneously with the Inspector Scan step, so that integration and security tests run at the same time.

Working in immutable infrastructure fashion one has full consistency between different environments. If

HELECLOUD

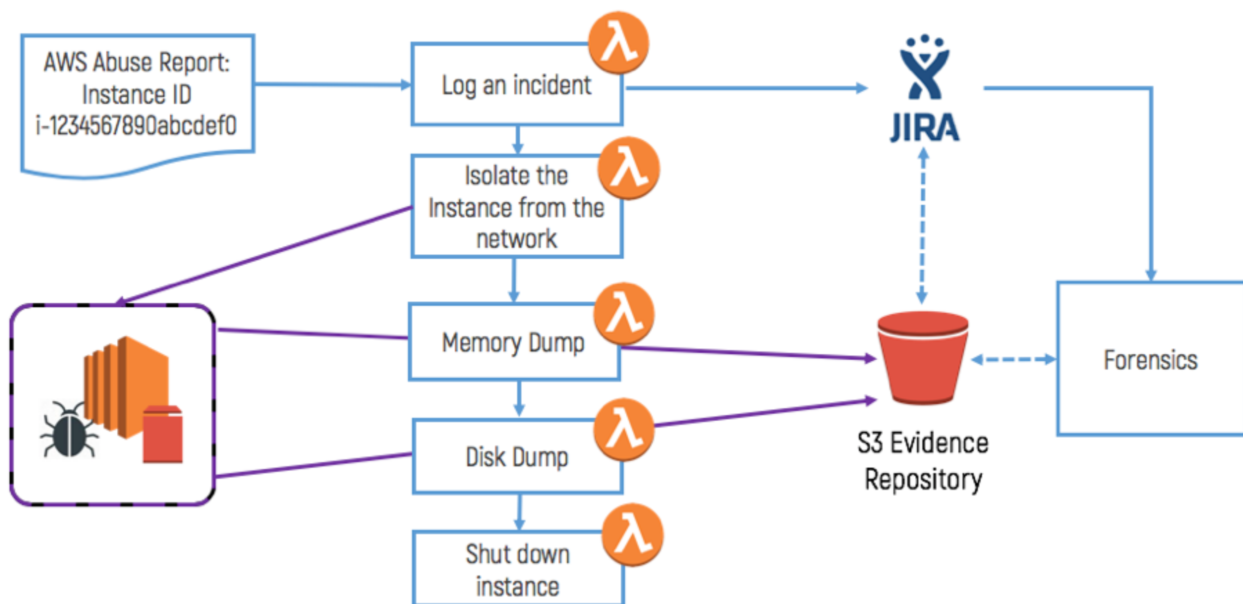
tests pass on one environment, the chance to pass in every other are really high, having in mind that the only difference is the application configuration.

For even faster feedback – Packer has a functionality to build AMIs, as well as to build Docker images – by using the same configuration file. This way the AMI and the Docker will be configured in the same way and the software installed will be the same.

In Packer terminology - the very same Packer Provisioners can be used with different Packer Builders.

Incident response automation on infected instance

Here is an example scenario, in which an instance is marked as infected – either by the internal (IDS/ GuardDuty) or external (AWS abuse report) source.



The whole process might be based on lambda functions or dedicated EC2 instance.

1) Instance is marked as infected
2) Incident is logged to a ticketing system and notification is send
3) Instance is isolated from the network by moving SecurityGroups to allow inbound connections from a whitelisted source
4) Memory dump is made and saved to S3 bucket– margarita shotgun might be used - https://margaritashotgun.readthedocs.io/en/latest/
5) EC2 Snapshot is made of the instance EC2 Volumes
6) Instance is shut down

For executing the whole process, a tool like AWS_IR might be used.

HELECLOUD

Conclusion

Working in immutable infrastructure and pre-build images guarantees that instances deployed in different accounts are the same bit by bit. Operating system, specific configuration of the OS, installed software with dependencies are all the same. The only thing that defers is the service configuration. This makes all the deployment easily reproducible. Automating AMI build and putting it in repository makes the whole infrastructure auditable and changes traceable.

Performing regular security scans, keeping results for audit purposes, proper tagging of the images and instances, notifying on issues found complements the security and compliance requirements. Scanning single instance of an image huge cost benefits can be achieved. AWS Inspector being a managed service makes it easy to integrate with the software delivery lifecycle. By Accelerate: State of DevOps 2018 "... and shifting left on security all positively contribute to continuous delivery."[4]

Having an image pre-baked makes the deployment process as fast as possible and it's perfect for fast scaling applications. Working with the same image on multiple environments guarantees full consistency at the extend that no such level can be achieved with configuration management tools.

Building AMI Factory as pipeline makes the process auditable and replicable, security scans and continuous delivery is achieved.

HELECLOUD

References

[1] https://www.allthingsdistributed.com/2016/03/10-lessons-from-10-years-of-aws.html

[2] https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c

[3] https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure

[4] https://devops-research.com/2018/08/announcing-accelerate-state-of-devops-2018/

Further readings

- https://aws.amazon.com/blogs/security/how-to-set-up-continuous-golden-ami-vulnerability-assessments-with-amazon-inspector/
- https://aws.amazon.com/blogs/awsmarketplace/announcing-the-golden-ami-pipeline/
- https://www.youtube.com/watch?v=4P_J3OiH42g - Using Amazon Inspector to Discover Potential Security Issues - AWS Online Tech Talks
- https://aws.amazon.com/blogs/aws/scale-your-security-vulnerability-testing-with-amazon-inspector/
- https://d1.awsstatic.com/whitepapers/managing-your-aws-infrastructure-at-scale.pdf
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-vulnerability_assessment
- https://aws.amazon.com/quickstart/architecture/compliance-hipaa/
- https://aws.amazon.com/quickstart/architecture/compliance-pci/
- "Release It!2.0" Michael T. Nygard
- "The Packer book" James Turnbull - https://packerbook.com