**AWS CSA-Pro Notes ACloudGuru Section 8 Deployment and Operations Management**

**Exam Tips:**

**Types of Deployments:**

**Understand the types of deployments and when each might be preferred in a given situation**

* **Rolling Deployment**
* **A/B Testing**
* **Canary**
* **Blue/Green**
  + These deployment styles really seem by preference of time and cost.
  + Rolling takes a long time, but does not cost much
  + A/B testing requires 2 full infrastructures which costs a lot, but is time effective
  + Canary seems to take the longest on knowing the effect of your new version
  + Blue/Green seems the best when you want to achieve immutability of your infrastructure, but costs the most
* Page: 5

**Know the various ways AWS can support Blue/Green deployments and when Blue/Green is not recommended**

* Blue/Green:
  + Update DNS with Route 53 to point to a new ELB or instance
  + Swap Auto-Scaling group already primed with new version instances behind the ELB
  + Change Auto-Scaling Group Launch Configuration to use new AMI version and terminate old instances
  + Swap environment URL of Elastic Beanstalk
  + Clone stack in AWS OpsWorks and update DNS
* Blue/Green not recommended:
  + Data store schema is too tightly coupled to the code changes
  + The upgrade requires special upgrade routines to be run during deployment
  + Off-the-shelf products might not be blue-green friendly

**Continuous Integration and Continuous Deployment:**

**Understand conceptually Continuous Integration, Continuous Delivery, and Continuous Deployment and their considerations**

* **Continuous Integration-** Merge code changes back to the main branch as frequently as possible with automated testing as you go
* **Continuous Delivery-** You have automated your release process to the point you can deploy at the click of a button
* **Continuous Deployment-** Each code changes that passes all stages of the release process is released to production with no human intervention required

**Know what AWS tools can be used to facilitate these methods of deployment**

* **AWS CodeCommit-** Hosted git repo
* **AWS CodeBuild-** Helps you compile code, run tests, and create deployment packages
* **AWS CodeDeploy-** Deploy deployment packages on EC2, Lambda, Elastic Beanstalk, ECS, or on-prem systems
* **AWS CodePipeline-** Orchestration mechanism that helps us do all these things together
* **AWS X-Ray-** Helps debug apps
* **AWS CodeStar-** Leverages all other previous services

**Elastic Beanstalk:**

**Know the components of Elastic Beanstalk and the platforms supported**

**Understand the deployment options with Elastic Beanstalk and the tradeoffs for each**

**CloudFormations:**

**Understand how CloudFormation delivers Infrastructure as Code and the benefits of that**

* Infrastructure as Code allows for repeatability in deployments of your infrastructure that can be used for testing or managing your infrastructure from one screen

**Elastic Container Service:**

**Know the difference between ECS and EKS – as well as the uniqueness of each**

* ECS:
  + Considered simpler to learn and use
  + ECS leverages AWS services like Route 53, ALB, and CloudWatch
  + Tasks- are instances of containers that are run on underlying compute but more or less isolated
  + Limited extensibility
* EKS:
  + Compatible with upstream K8s so it is easy to lift and shift from other K8s
  + Considered more feature-rich and complex with a steep learning curve
  + A hosted K8s platform that handles many things internally
  + Pods- are containers collocated with one another and can have shared access to each other
  + Extensible via a wide variety of third-party and community add-ons

**Understand the difference between EC2 Launch Types and Fargate Launch Types**

* EC2 Launch type:
  + You explicitly provision EC2 instances
  + You are responsible for upgrading, patching, care of EC2 pool
  + You must handle cluster optimization
  + More granular control over infrastructure
* Fargate Launch type:
  + The control plane asked for resources and Fargate automatically provisions them
  + Fargate provisions compute as needed
  + Fargate handles cluster optimization
  + Limited control, as infrastructure is automated

**API Gateway:**

**Understand what (and how) you would deploy an API on API Gateway**

**Remember that API Gateway is designed to serve up REST APIs**

**Management Tools:**

**Know when and what you can expect when using AWS Config. Understand the purpose of a Config Rule**

* AWS Config is a management tool used to monitor your whole infrastructures configuration on a historical basis
* You can use pre-made or custom Config rules to monitor certain compliance standards you want to keep (example- Is CloudTrail on) If it is not on there can be an email sent to your team so you can be notified of non-compliant resources

**Know the different flavors of AWS OpsWorks and, conceptually, what Chef and Puppet offer**

* **OpsWorks for Chef Automate:** Fully managed implementation of Chef
* **OpsWorks for Puppet Enterprise:** Fully managed implementation of Puppet
* **OpsWorks Stacks:** An AWS creation and uses an embedded Chef solo client installed on EC2 instances to run Chef recipes

**Understand the difference between AWS OpsWorks Stacks and AWS Ops Works for Chef Automate**

**Remember that OpsWorks is a global service, but you can only manage resources in the region you created the OpsWorks stack**

**System Manager:**

**Know the various services under the System Manager heading and how they help simplify management of larger landscapes**

* Page: 10
* Systems Manager can help with basic system administration tasks in a variety of ways into a large fleet of instances without having to SSH into each individually

**Can manage both AWS-based and on-prem systems so long as they are supported Oss**

**Understand Patch Manager pre-defined baselines and that they act as a pre-approval gatekeeper**

**Understand the various SSM document types and their purposes**

**Command Document:**

* Used with Run Command or State Manager
* Run Command uses command documents to execute commands. State Manager uses command documents to apply a configuration.
* These actions can be run on one or more targets at any point during the lifecycle of an instance

**Policy Document:**

* Used with the State Manager
* Policy documents enforce a policy on your targets. If the policy document is removed, the policy action (for example, collection inventory) no longer happens

**Automation Document:**

* Used with Automation
* Use automation documents when performing common maintenance and deployment tasks such as creating or updated and AMI

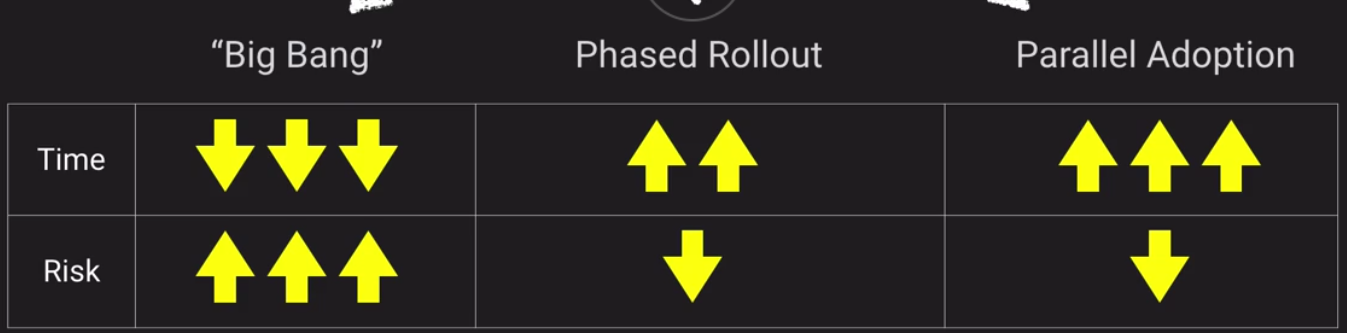
**Types of Deployments:**

**Software Deployments:**

**Big Bang-** Everything goes live all at once

**Phased Rollout-** Gradual overtime deploy

**Parallel Adoption-** Implement the new while still using the old

****

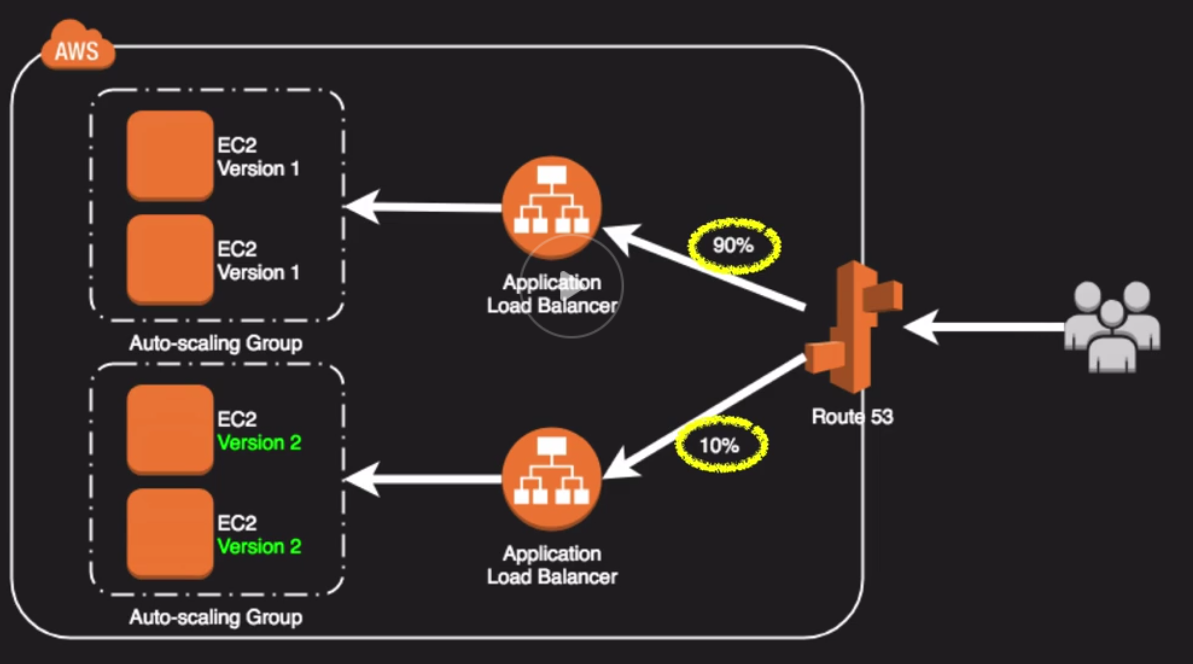
**Phased Rollout Types with AWS:**

**Rolling Deployment:**

* You have an ELB with an Auto-Scaling group managing your EC2 instances
* Create new Launch Configuration with the updated AMI
* Start terminating old EC2 instances
* New instances will be launched with version 2 AMI

**A/B Testing:**

* Use Route 53
* Have 2 infrastructure built the same way. ELB’s behind both with Auto-Scaling groups for both.
  + Only difference would be having 1 Auto-Scaling group deploying version 2
* Use Route 53 Weighted Round Robin to send most traffic to the old version and a little traffic to the new version and wait to see if anything goes wrong
* Overtime after it is tested we can shift 100% of traffic to go to version 2 and 0 to version 1



**Canary Release:**

* Deploy version 2 into production on a small amount of instances
* Wait and see if you have any errors before deploying to the rest of the servers

**Blue-Green Deployment:**

* 1 section of application running version 1 with ELBs and Auto-Scaling groups
* Create a second VPC with an ELB and Auto-Scaling groups running version 2
* When ready repoint Route 53 to the new version
* If you have problems you can just repoint Route 53 back

The goal of blue/green deployments is to achieve immutable infrastructure, where you don’t make changes to your application after it’s deployed, but redeploy altogether.

**Blue-Green Methods:**

* Update DNS with Route 53 to point to a new ELB or instance
* Swap Auto-Scaling group already primed with new version instances behind the ELB
* Change Auto-Scaling Group Launch Configuration to use new AMI version and terminate old instances
* Swap environment URL of Elastic Beanstalk
* Clone stack in AWS OpsWorks and update DNS

**Blue-Green Contraindications:**

* Data store schema is too tightly coupled to the code changes
* The upgrade requires special upgrade routines to be run during deployment
* Off-the-shelf products might not be blue-green friendly

**Continuous Integration and Continuous Deployment:**

**CI, CD, and CD too:**

**Continuous Integration-** Merge code changes back to the main branch as frequently as possible with automated testing as you go

**Continuous Delivery-** You have automated your release process to the point you can deploy at the click of a button

**Continuous Deployment-** Each code changes that passes all stages of the release process is released to production with no human intervention required

**CI/CD Considerations:**

* Objective is to create smaller, incremental compartmentalized improvements and features
* Lowers deployment risk and tries to limit negative impact
* Test automation game must be strong
* Feature toggle patterns useful for dealing with in-progress features not ready for release (versus more traditional branching strategies)
* Microservice architectures lend themselves well to CI/CD practices

**AWS Development Lifecycle Tools:**

**AWS CodeCommit-** Hosted git repo

**AWS CodeBuild-** Helps you compile code, run tests, and create deployment packages

**AWS CodeDeploy-** Deploy deployment packages on EC2, Lambda, Elastic Beanstalk, ECS, or on-prem systems

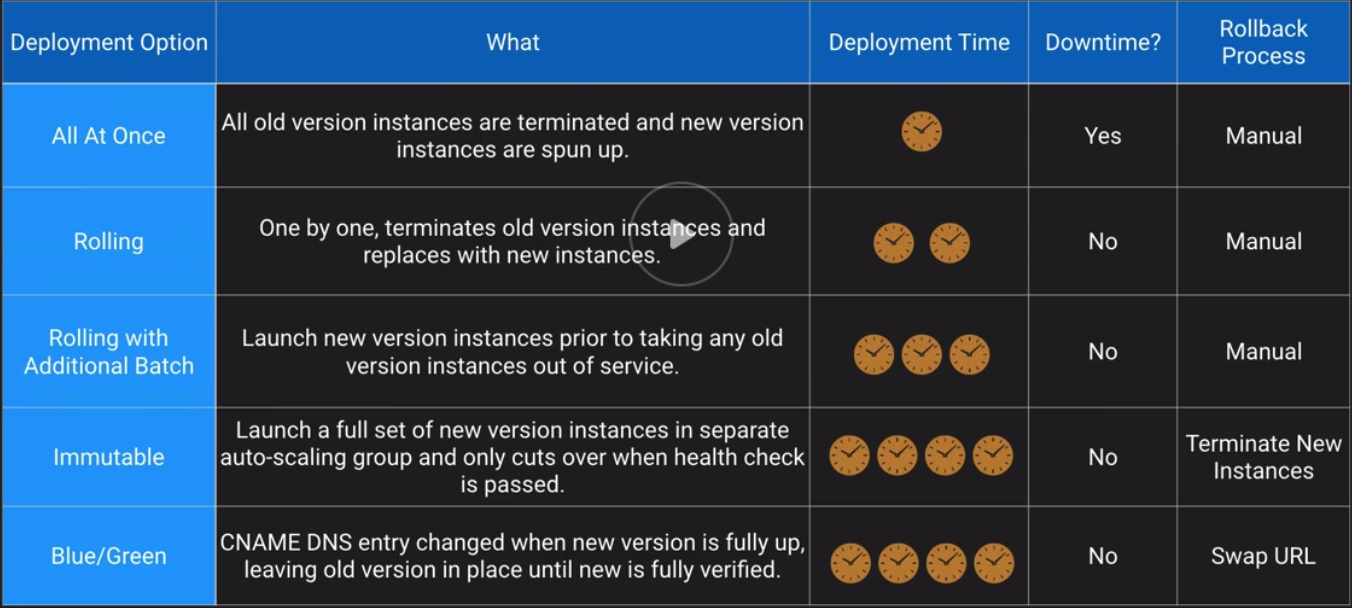
**AWS CodePipeline-** Orchestration mechanism that helps us do all these things together

**AWS X-Ray-** Helps debug apps

**AWS CodeStar-** Leverages all other previous services

**Elastic Beanstalk:**

* Orchestration service to make I push-button easy to deploy scalable web landscapes
* Wide rand of supported platforms – from Docker to PHP to Java to Node.js
* Multiple environments within Apps
* Great for ease of deployment, but not great if you need lots of control and flexibility

****

**CloudFormation:**

* Infrastructure as Code
* Using JSON or YAML, you can model and provision entire landscapes
* Repeatable, automatic deployments and rollbacks
* Nest common components for reusability
* Supports over 300 Resource Types
* Supports custom resources via SNS or Lambda

**CloudFormation Concepts:**

**Template-** The JSON or YAML test files that contain the instructions for building-out the AWS environment

**Stacks-** The entire environment described by the template and created, updated, and deleted as a single unit

**Change Sets-** A summary of proposed changes to your stack that will allow you to see how those changes might impact your existing resources before implementing them

The only section in CloudFormations is the Resource sections.

**Stack Policies:**

* Protect specific resources within your stack from being unintentionally deleted or updated
* You can add Stack Policies via the console or CLI when creating a stack
* After creation you can only add and edit via the CLI
* Once applied you cannot remove a Stack Policy
* By default, they protect everything

**CloudFormation Best Practices:**

* AWS provides Python “helper scripts” which can help you install software and start services on your EC2 instances
* Use CloudFormation to make changes to your landscape rather than going directly into the resources
* Make use of Change Sets to identify potential trouble spots in your updates
* Use Stack Policies to explicitly protect sensitive portions of your stack
* User a version control systems to track changes to templates

**Elastic Container service:**

* Both are highly available, highly scalable container platforms

**Amazon ECS:**

AWS specific platform that supports Docker containers

* Considered simpler to learn and use
* ECS leverages AWS services like Route 53, ALB, and CloudWatch
* Tasks- are instances of containers that are run on underlying compute but more or less isolated
* Limited extensibility

**Amazon EKS:**

* Compatible with upstream K8s so it is easy to lift and shift from other K8s
* Considered more feature-rich and complex with a steep learning curve
* A hosted K8s platform that handles many things internally
* Pods- are containers collocated with one another and can have shared access to each other
* Extensible via a wide variety of third-party and community add-ons

**Launch Types for ECS:**

**EC2:**

* You explicitly provision EC2 instances
* You are responsible for upgrading, patching, care of EC2 pool
* You must handle cluster optimization
* More granular control over infrastructure

**Fargate:**

* The control plane asked for resources and Fargate automatically provisions them
* Fargate provisions compute as needed
* Fargate handles cluster optimization
* Limited control, as infrastructure is automated

**API Gateway:**

* Managed, high availability service to front-end REST APIs
* Backed with custom code via Lambda, as a proxy for another AWS Service or any other HTTP API on AWS or elsewhere
* Regionally based, private or edge optimized
* Can be privately available
* Supports API Keys and Usage Plans for user identification, throttling or quota management
* Using CloudFront behind the scenes and custom domains and SNI are supported
* Can be published as products and monetized on AWS Marketplace

**API Caching:**

* Cache responses from your backend
* Helps with performance and DynamoDB load

**Management Tools:**

**AWS Config:**

* Allows you to assess, audit and evaluate configurations of your AWS resources
* Very useful for configuration management as part of an ITIL program
* Creates a baseline of various configuration settings and files then can track variations against that baseline
* AWS Config Rules can check resources for certain desired conditions and if violations are found, the resources is flagged as “noncompliant”
  + Config rule examples: Is Backup enabled on RDS
  + Is CloudTrail enabled on the AWS account
  + Are EBS volumes encrypted

**AWS OpsWorks:**

* Managed instance of Chef and Puppet-2 very popular automation platforms
* Provide configuration management to deploy code, automate tasks, configure instances, perform upgrades

**OpsWorks Options:**

**OpsWorks for Chef Automate:** Fully managed implementation of Chef

**OpsWorks for Puppet Enterprise:** Fully managed implementation of Puppet

**OpsWorks Stacks:** An AWS creation and uses an embedded Chef solo client installed on EC2 instances to run Chef recipes

Supports on-prem servers as well with an agent install

**Stacks-** Are collection of resources needed to support a service or application

**Layers-** Represent different components of the application delivery hierarchy

* Layer Examples: EC2 instances
* RDS instances
* ELBs

Stacks can be cloned – but only within the same region

OpsWorks is a global service. Stacks are regional

**AWS System Manager:**

* Centralized console and toolset for a wide variety of system management tasks
* Designed for managing a large fleet of systems-tens or hundreds
* SSM Agent enables System Manager features and support all OSs supported by OS as well as back to windows server 2003 and Raspbian
* SSM Agent installed by default on recent AWS-provided base AMIs for Linux and Windows
* Manages AWS-based and on-prem systems by installing the agent

**Services:**

**Inventory:**

* Collect OS, application and instances metadata about instances
* Which instances have Apache HTTP Server 2.2.x or earlier

**State Manager:**

* Creates states that represent a certain configuration that is applied to instances
* Keep track of which instances have been updated to the current stable version of Apache HTTP Server

**Logging:**

* CloudWatch Log agent and stream logs directly to CloudWatch from instances
* Stream logs of our web servers directly to CloudWatch for monitoring and notifications

**Parameter Store:**

* Shared secure storage for config data, connection strings, passwords, etc
* Store and retrieve RDS credentials to append to a config file upon boot

**Insights Dashboard:**

* Account-level view of Cloudtrail, Config, and Trusted Advisor
* Single viewport for any exceptions on config compliance

**Resource Groups:**

* Group resources through tagging for organization
* Create a dashboard for all assets belonging to our prod ERP landscape

**Maintenance Windows:**

* Defines schedules for instances to patch, update apps, run scripts and more
* Define off hours as a Window for updating

**Automation:**

* Automating routine maintenance tasks and scripts
* Stop DEV and QA instances every Friday and restart Monday morning

**Run Command:**

* Run commands and scripts without logging in via SSH or RDP
* Run a shell script on 53 different instances at the same time

**Patch Manager:**

* Automates process of patching instances for updates
* Keep a fleet at the same patch level by applying new security patches during next Maintenance Window

**Patch Manager Baseline:**

* Defines which patches are auto approved
* AWS has default baselines or you can create your own

**AWS System Manager Documents:**

**Command Document:**

* Used with Run Command or State Manager
* Run Command uses command documents to execute commands. State Manager uses command documents to apply a configuration.
* These actions can be run on one or more targets at any point during the lifecycle of an instance

**Policy Document:**

* Used with the State Manager
* Policy documents enforce a policy on your targets. If the policy document is removed, the policy action (for example, collection inventory) no longer happens

**Automation Document:**

* Used with Automation
* Use automation documents when performing common maintenance and deployment tasks such as creating or updated and AMI