**Advanced Testing Practices Using AWS DevOps Tools (Resume)**

**Module 1: Testing Overview**

**Summary:** One of DevOps’ key features is the ability to integrate testing and  security into the development cycle far earlier into the process. Testing runs into the snags before they become too ingrained to easily repair. We discuss our 4 main tests - unit tests, service/integration tests, performance/compliance tests and UI/end-to-end tests. Unit tests isolate a particular aspect. S/I tests compare the connection between components. P/C tests diagnose the overall application performance and stability. UI/E tests look at the burden placed on users.

One way to make testing far more manageable is to automate as many tests as possible. Automated deaths make life easier for developers by moving a lot of mental energy when putting code together from the developer to the program itself. This saves time and makes things more consistent.

We go through the testing pyramid, which shows the order of speed and cost for each test. Test coverage should also be considered. If your tests are looking for redundant or irrelevant material, then you need to improve your test coverage.

**Core Technologies: /**

**Module 2: DevOps Review**

**Summary:** We review how to use our AWS tools to construct a DevOps compliant environment. DevOps in general is a methodology that specializes in compartmentalizing projects into small, rapid to code sections and then shooting them down an optimized pipeline and centralized repository.

We review our tools as well. CodeCommit works as a Git-based repository service to keep the versions from different teams integrated. CodeBuild manages your software packages and most of your testing. CodeDeploy handles your deployments and keeps your control panel centralized. CodePipeline is an overall service that puts it all together and lets you control your releases. We discuss a little about serverless and third party options for checking out our code as well.

**Core Technologies:** AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline

**Module 3: Continuous Integration**

**Summary:** We go over test-driven-development as we cover the continuous integration aspect of the pipeline in AWS. It is critical that developers stay on the same live build before they spiral out of control, which is where AWS tools come in to run checks and tests on code commits. Code standards are usually implemented at the company-level to ensure everyone notarizes and codes in similar practices. Automatic code analysis is a supplemental tool used to prevent code degradation.

Linters are used to clean up code automatically as another tool for developers to save time. Linters aren’t tests per se but they do check for a lot of syntax errors and loose validations. Testing itself is often written ahead of time to review for intended outputs.

Test-driven-development and behavior-driven-development should both be considered when creating your code. The former handles the raw functionality while the latter considered the human expectations of what should be outputted. We also review some testing frameworks that are used for implementing tests.

Commits should be done frequently and based from the master branch in a DevOps system. CodeBuild is also quite handy for storing build state and test reporting.

**Core Technologies:** AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, AWS CodePipeline, JUnit, PyTest, PHPSpec, AWS CloudFormation, Amazon CodeGuru

**Module 4: Continuous Delivery**

**Summary:** We begin with going from unit testing to functional testing, which works at the component level. Regression testing comes afterwards and coordinates the software to ensure nothing conflicts with action at other sectors. Performancing testing moves from simple functionality to overall benchmarks.

CodePipeline and BlazeMeter are performance focused frameworks that can streamline the testing process.

User acceptance testing is like end-to-end tests in that it incorporates user experience and expectations into the testing rubric.

Synthetic testing tries to anticipate pathways and logics that users will take and ensure that those command lines work as intended.

For security purposes, DevOps splinters into DevSecOps which focuses primarily on the security of the CI/CD pipeline.

**Core Technologies:** AWS Device Farm, AWS CodePipeline, BlazeMeter, Apache JMeter

**Module 5: Continuous Deployment**

**Summary:** After Continuous Delivery, developers need to parse and approve of the changes that have been pushed into the pipeline. This is where Continuous Deployment comes in. Rolling deployments have zero downtime and have multiple versions of code running in different sectors. Works very quickly, but can be prone to complexity and speed issues.

Developers use health checks to see where a deployment is working successfully. CodeDeploy helps a lot to remove some of the manual work in putting together configurations and health checks. CodeDeploy works with in-place deployment and blue/green deployment methods.

The Appsec file in CodeDeploy is used to manage lifecycle hooks which are locations that outline functions or scripts for testing and deployment in an application.

Segmented and canary deployments try to mitigate risks of deployment failure. Canary deployments additionally add live production traffic as an additional measure. Synthetic tests like these can simulate live experience even without an active base to test with.

At the end of the pipeline are manual and automatic approvals. Manual approvals are done by developers while automatic approvals are done through a predetermined set of guidelines.

You can also close the pipeline, halt promotions or even issue rollbacks from the AWS Management Console as you need.

**Core Technologies:** AWS CodeDeploy, Appsec, AWS Lambda, Amazon CloudWatch Synthetics, AWS Identity and Access Management (IAM), AWS CLI