DevOps Essentials DevOps Concepts and Practices Notes

Build Automation

What is Build Automation?

Build automation- is automation of the process of preparing code for deployment to a live environment.

* Depending on what languages are used, code needs to be compiled, linted, minified, transformed, unit tested, etc.
* Build automation is taking these steps and automating them to make sure these steps are done in a consistent way. This is usually done using a build script or a build tool.
* The tools of build automation often differ depending on what programming languages and frameworks are used, but they all have one thing in common: Automation!

What does build automation look like?

* Usually, build automation looks like running a command-line tool that builds code using config files and/or scrips that are treated as part of the source code.
* Build automation is independent of an IDE. (IDE=integrated development environment)
* Even if you can build within the IDE, it should be able to work the same way outside of the IDE.
* As much as possible, build automation should be agnostic of the configuration of the machine that it is built on.
* Your code should be able to build on someone else’s machine the same way it builds on yours

Why do build automation?

* Build automation is fast – Automation handles tasks that would otherwise need to be done manually.
* Build automation is consistent – The build happens the same way every time, removing problems and confusion that can happen with manual builds.
* Build automation is repeatable – The build can be done multiple time with the same result. Any version of the source code can always be transformed into deployable code in a consistent way.
* Build automation is portable – The build can be done the same way on any machine. Anyone on the team can build on their machine, as well as on a shared build server. Building code doesn’t depend on specific people or machines.
* Build automation is more reliable – There will be fewer problems caused by bad builds.

Build automation is a developer centric concept., but understanding how important build operations is to the development teams you are working with is important.

Continuous Integration

What is Continuous Integration?

* Continuous Integration (CI): the practice of frequently merging code changes done by developers.
* Traditionally, developers would work separately, perhaps for weeks at a time, and then merge all of their work together at the end in one large effort.
* Continuous integration means merging constantly throughout the day, usually with the execution of automated tests to detect any problems caused by the merge.
* Merging all the time could be a lot of work, so to avoid that is should be automated.

What does continuous integration look like?

* Continuous integration is usually done with the help of a CI server.
* When a developer commits a code change, the CI server sees the change and automatically performs a build, also executing automated tests.
* This occurs multiple times a day
* If there is any problem with the build, the CI server immediately and automatically notifies the developers.
* If anyone commits code that “breaks the build” they are responsible for fixing the problem or rolling back their changes immediately so that other developers can continue working.

Why do Continuous Integration?

* Early detection of certain types of bugs – If code doesn’t compile or an automated test fails, the developers are notified and can fix it immediately. The sooner these bugs are detected the easier they are to fix.
* Eliminate the scramble to integrate just before a big release – the code is constantly merged, so there is no need to do a big merge at the end.
* Makes frequent releases possible – Code is always in a state that can be deployed to production.
* Makes continuous testing possible – Since the code can always be run, QA testers can get their hands on it all throughout the development process, not just at the end.
* Encourages good coding practices – Frequent commits encourages simple, modular code.

Continuous Delivery and Continuous Deployment

What is Continuous Delivery?

* Continuous Delivery (CD): the practice of continuously maintaining code in a deployable state.
* Regardless of whether or not the decision is made to deploy, the code is always in a state that is able to be deployed.
* Some use the terms continuous delivery and continuous deployment interchangeably, or simply use the abbreviation CD. They are similar but separate concepts.

What is continuous deployment?

* Continuous deployment: The practice of frequently deploying small code changes to production.
* Continuous delivery is keeping the code in a deployable state. Continuous deployment is actually doing the deployment frequently.
* Some companies that do continuous deployment deploy to production multiple times a day.
* There is no standard for how often you should deploy, but in general the more often you deploy the better.
* With continuous deployment, deployments to production are routine commonplace events.

What does Continuous Delivery and Continuous Deployment look like?

* Each version of the code goes through a series of stages such as automated build, automated testing, and manual acceptance testing. The result of this process is an artifact or package that is able to be deployed.
* When the decision is made to deploy, the deployment is automated.
* If a deployment causes a problem, it is quickly and reliably rolled back using an automated process.
* Rollbacks aren’t a big deal because the developers can redeploy a fixed version as soon as they have one available.

Why do continuous delivery and continuous deployment?

* Faster time-to-market – Get features into the hands of customers more quickly rather than waiting for a lengthy deployment process that doesn’t happen often.
* Fewer problems cause by the deployment process – Since the deployment process is frequently used, any problems with the process are more easily discovered.
* Lower risk – The more changes are deployed at once, the higher the risk. Frequent deployments of only a few changes are less risky.
* Reliable rollbacks – Robust automation means rollbacks are a reliable way to ensure stability for customers, and rollbacks don’t hurt developers because they can roll forward with a fix as soon as they have one.
* Fearless deployments – Robust automation plus the ability to rollback quickly means deployments are commonplace, everyday events rather than big, scary events.

Infrastructure as Code

What is Infrastructure as Code?

* Infrastructure as Code (IaC) : manage and provision infrastructure through code and automation.
* With Iac, instead of doing things manually, you use automation and code to create and change: servers, instances, environments, containers, and other infrastructure.

What does IaC look like?

* Without IaC you might: ssh into a host, issue a series of commands to perform the change.
* With IaC: instead change some code or configuration files that can be used with an automation tool to perform changes, commit them to source control, then use an automation tool to enact the changes defined in the code and/or configuration files.
* With IaC, provisioning new resources and changing existing resources are both done through automation.

Why do IaC?

* Consistency in creation and management of resources – The same automation will run the dame way every time.
* Reusability – Code can be used to make the same change consistently across multiple hosts and can be used again in the future.
* Scalability – Need a new instance? You can have on configured exactly the same way as the existing instances in minutes (or seconds).
* Self-documenting – With IaC, changes to infrastructure document themselves to a degree. The way a server is configured can be viewed in source control, rather than being a matter of who logging in to the server and did something.
* Simplify the complexity - Complex infrastructures can be stood up quickly once they are defined as code. A group of several interdependent servers can be provisioned on demand.

What is Configuration Management?

* Configuration Management: Maintaining and changing the state of pieces of infrastructure in a consistent, maintainable, and stable way. Automated management of infrastructure configuration.
* Changes always need to happen – configuration management is about doing them in a maintainable way. Keep infrastructure up to date and meet changing business needs.
* Configuration management allows you to minimize **configuration drift** – the small changes that accumulate over time and make systems different from one another and harder to manage.
* IaC is very beneficial for configuration management

What does configuration management look like?

* You need to upgrade a software package on a bunch of servers: Without good configuration management, you log into each server and perform the upgrade. However, this can lead to a lot of problems. Perhaps one server was missed due to poor documentation, or perhaps something doesn’t work while the versions are temporarily mismatched between servers, causing a lot of downtime while you do the upgrade.
* With good configuration management, you define the new version of software package in a configuration file or tool and automatically roll out the change to all of the servers.
* Configuration management is about managing your configuration somewhere outside of the servers themselves.

Why do configuration management?

* Save time – It takes less time to change the configuration.
* Insight – With good configuration management, you can know about the state of all pieces of a large and complex infrastructure.
* Maintainability – A more maintainable infrastructure is easier to change in a stable way.
* Less configuration drift – It is easier to keep a standard configuration across a multitude of hosts.

Orchestration

What is orchestration

* Orchestration: automation that supports processes and workflows, such as provisioning resources.
* With orchestration, managing a complex infrastructure is less like being a builder and more like conducting an orchestra.
* Instead of going out and creating a piece of infrastructure, the conductor simply signals what needs to be done and the orchestra performs it.
  + The conductor does not need to control every detail
  + The musicians (automation) are able to perform their piece with only a little bit of guidance

What does orchestration look like?

* A customer requests more resources for a web service that is about to see a heavy increase in usage due to a planned marketing effort.
* Instead of manually standing up new nodes, operation engineers use an orchestration tool to request five more nodes to support the service.
* A few minutes later, the tool has five new nodes up and running
* A monitoring tool detects an increased load on the service, an orchestration tool responds to this by spinning up additional resource to handle the load, when the load decreases again, the tool spins the additional resources back down, freeing them up to be used by something else, this is auto scaling in a nut shell. No human interaction needed after the initial configuration.

Why do orchestration?

* Scalability – Resources can be quickly increased or decreased to meet changing needs.
* Stability – Automation tools can automatically respond to fix problems before users see them. A node or instance becomes unhealthy your orchestration tool will notice and fix it.
* Save time – Certain tasks and workflows can be automated, freeing up engineers ‘ time.
* Self- service – orchestration can be used to offer resource to customer in a self-service fashion. AWS is purely automation using orchestration, when you request an instance it is auto given to you without humans due to their orchestration.
* Granular insight into resource usage – Orchestration tools give greater insight into how many resources are being used by what software, service, or customers.

Monitoring

What is monitoring?

* Monitoring: The collection and presentation of data about the performance and stability of services and infrastructure. Allows you to respond to problems cause by frequent deployments.
* Monitoring tools collect data over things such as: Usage of memory, CPU, disk i/o, other resources over time, Application logs, Network Traffic, ETC.
* The collected data is presented in various forms, such as charts and graphs, or in the form of real-time notifications about the problems.

What does monitoring look like?

* Real-time notifications: Performance on the website is beginning to slow down, a monitoring tool detects that response time are growing, and an administrator is immediately notified and is able to intervene before downtime occurs.
* Postmortem analysis: Something went wrong in production last night, it is working now, but we don’t know what caused it. Luckily, monitoring tools collected a lot of data during the outage. With that data, developers and operation engineers are able to determine the root cause and fix it.

Why do monitoring?

* Fast recovery – The sooner a problem is detected, the sooner it can be fixed. You want to know about a problem before your customer does.
* Better root cause analysis – The more data you have, the easier it is to determine the root cause of a problem.
* Visibility across teams – Good monitoring tools give useful data to both developers and operations people about the performance of code in production.
* Automated response – Monitoring data can be used alongside orchestration to provide automated responses to events, such as automated recovery from failures.

Microservices

What are microservices?

* Microservices: a microservice architecture breaks an application up into a collection of small, loosely-coupled services.
* Traditionally, apps used a monolithic architecture. In a monolithic architecture, all features and services are part of one large application.
* Microservices are small: each microservice implements only a small piece of an application’s overall functionality.
* Microservices are loosely coupled: Different microservices interact with each other using stable and well-defined API’s. this means that they are independent of one another.

What do microservices look like?

* There are many different ways to structure and organize a microservice architecture
* For example, a pet shop application: needs a pet inventory service, customer details service, authentication service, pet adoption request service, and a payment processing service.
* Each one of these is its own codebase and a separate running process. They can all be built, deployed and scaled separately.

Why use microservices?

* Modularity – Microservices encourage modularity. In monolithic apps, individual pieces become tightly coupled, and complexity grows. Eventually, it is very hard to change anything without breaking something.
* Technological flexibility – You don’t need to use the same languages and technologies for every part of the app. You can use the best tool for each job.
* Optimized scalability – You can scale individual parts of the app based upon resource usage and load. With a monolith, you have to scale up the entire application, even if only one aspect of the service actually needs to be scaled.
* Microservices aren’t always the best choice. For smaller, simpler apps a monolith might be easier to manage.