Gitops: The clean way to manage infrastructure

- Yashodhan Ghadge

Outline

Lecture outline

- Traditional ways to manage infrastructure
- Security, infrastructure drift, compliance view
- GitOps: how to manage it cleanly
- Security, drift , compliance view
- Live Hands On demo
- QA
- Closing remarks

Terminology

- SSH - secure shell, a command line utility that allows to connect to remote computers over the internet
- CI/CD - continuous integration , continuous deployment
- SCP - secure copy - like the linux cp commandline tool but works over SSH
- Docker - a relatively new sandboxed container that sandboxes a mini OS that carries the dependencies and code.
- OCI - open container initiative - like an RFC document so that containers can interoperate on all supporting platforms
- VM - virtual machine
- VPC - virtual private cloud - an isolated network environment in the cloud
- WAF - web application firewall

Terminology

- PII - personally identifiable information

Traditional methods

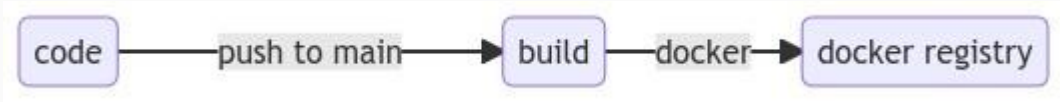
Ways to deploy

Exploring the different ways where deploys take place across the organization.

Shown on the right is the Standard Deployment pipeline.

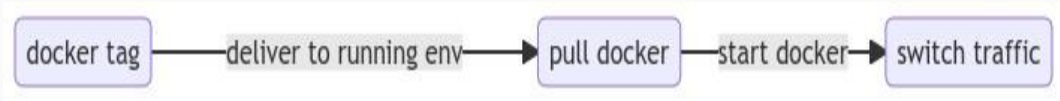
Code is built only once, and for each new environments we switch the environment variables

Code build

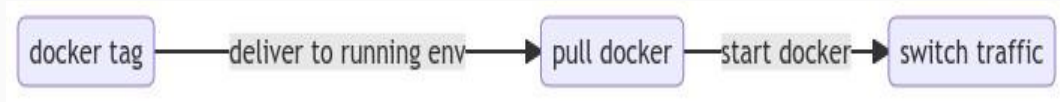


Deployment Strategy

Env 1



Env 2



Methods to execute deployment strategies

When it comes to physically inserting the new code into a running environment, there are 3 main strategies that are used.

These strategies are used widely across many many organizations that have been around for 5+ years.

We will cover the pros and cons of each of these methods to understand why we need a new more transparent procedure.

Deployment strategies

- SSH based deploy
- Pull based
- Manual uploading

SSH based deploy

SSH based deploys have a simple straight-forward mechanism. There is a public key present on the live server and either the devops team assigns an employee or runs an automation (CI/CD, bash script, etc) to to SCP (copy over SSH) the code (can be dockerfiles, code files, zips,etc etc) to the server.

This procedure is pretty much how humans connect to machines across the internet. So this becomes the most obvious choice of deploying

SSH based deploy (contd)

Pros

- Simple to deploy, requires only the payload, a bash script or instructions to stop old code and start new code.
- Servers can be fine tuned to include monitoring, logs collection, health checks
- Static IPs increase confidence when used by external application in high security applications like banking transaction API servers, file servers where legal documents are stored, govt projects related data. Since they can add that IP to allowlists.

SSH based deploy (contd)

Cons

- Servers are not upgraded for long periods of time. Leading to increased vulnerabilities.
- Cloud based servers have it difficult to assign static IPs, and need usage of cloud vendor APIs to determine the IP using tags/ids etc
- Private keys need to be secured and access to the VM has to be tightly controlled. Often, organizations lose track of who has been given these keys and often employees who leave the organization still retain access to these servers and have known to cause damages
- Code and server become highly coupled and it increases legacy dependencies and cost of maintenance increases drastically as time goes on
- If a server goes down, it leads to direct downtimes, even if it is a load balanced environment, as failures increase loads on other servers which often fail under increased stress

SSH based deploy (contd)

Cons (contd)

- Difficult to scale

Pull based deploys

This method has a simple bash script or a bespoke program that is inserted in the running VMs. This program/ script has a simple job. It is configured to either read databases like redis, sql, consul, or expose an API to which one can communicate. The payload this reads or is sent is the generally the new version of the code that is to be deployed. This is the main concept behind even modern deployment strategies like gitOps and kubernetes.

Once this program/script detects that a new version has been setup, it will execute the instructions that its programmed to perform. A simple example can be to read a redis key, once its value updates, pull the docker image related to the new value in the redis key, and deploy the new docker and stop the older running docker.

Pull based deploys(contd)

Pros

- Cleanly deploy new version across multiple servers
- Servers can be managed independently
- Requires a deployment of the agent periodically, hence lowers the dependency on SSH and private keys. Most times, new agent versions are baked into new VM images, and the whole VM itself is recreated.
- The DB / Api call logs provide a clean history of previous versions, dates, easier for audit trails
- Easier to blur the boundaries between an operations team and a development team, as even the development team can push new versions

Pull based deploys(contd)

Pros

- Easier to roll back (simply update the DB / Api call with the older version’s commit SHA or release version)
- Easier to setup blue green and canary deployments.

Pull based deploys(contd)

Cons

- Writing this agent is a tough task, lot of code to ensure that correct versions are being pulled, an explicit mechanism to diff the current deployed version vs the new version is required
- The agent, if vulnerable, makes the whole fleet of servers vulnerable
- Agent updates are hard to propagate, any bugs might block deployments
- Requires a dedicated development effort for the agent, and more infrastructure like the redis database, which increases cost

Manual uploading

This is a catch all process, and encompasses a lot of different scenarios, like physical approval forms , compliance based checks, deployment to high security environments, etc.

This process is mainly used when the systems are bespoke or too small scale or in high security environments, where the servers are air gapped.

Security, Compliance, and drift

- While running any codebases, there are always dual security modes, also known as the principle of shared responsibility. With this, the application development teams are responsible for the security of the application and the data it generates. While the devOps teams are responsible for security of the infrastructure.
- While the application teams work on the app security, we will focus our attention on how to achieve security of the infrastructure. Broadly, security of infrastructure is divided into 3 major categories
    - Network layer security - port restrictions, VPC peering restrictions, IP allow/block lists, WAF, etc
    - Hardware level security - patching OS and kernel, programs , databases, etc
    - Access level security - maintaining access audit trails, having processes in place to allow/ revoke individual access to the resources, read write protection of PII data

Security, Compliance, and drift (contd)

- Compliance, which is a set of mandatory rules and processes, require that each of the previous security policies be enforced and followed with an audit trail
- Drift is detecting whether or not, our deployed environment is running as expected or has it drifted away, ie, some characteristics have changed or not, if the configuration we applied is still running as expected, and if the capacity we provisioned for is in line with expected traffic.
- When these 3 factors come together, in a traditional deployment setting, we have difficulties with the accesses, because maintaining a list of who has and who hasn't been given access to is hard, maintaining inventory of the exact configuration of VMs can be pretty hard, and finally the setting up of drift detection is almost impossible due to the common practices of intermixing manual and automated workflows along with a lack of proper transparency.

GitOps

GitOps

- GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.
- It works with the system rather than against it, and can be easily adopted in any scenarios and regardless of the size of organizations.
- Modern applications are developed with speed and scale in mind. Organizations with a mature DevOps culture can deploy code to production hundreds of times per day. DevOps teams can accomplish this through development best practices such as version control, code review, and CI/CD pipelines that automate testing and deployments.

GitOps

- GitOps is used to automate the process of provisioning infrastructure. Similar to how teams use application source code, operations teams that adopt GitOps use configuration files stored as code (infrastructure as code). GitOps configuration files generate the same infrastructure environment every time it’s deployed, just as application source code generates the same application binaries every time it’s built.

GitOps: what a repo looks like

- To get started with gitops 3 things are required.
  - 1. A repository that will contain your changes
  - 2. Tools that will connect and read the repo created in step 1 to execute the instructions within
  - 3. Access controls to tools, and environment definition for running the tools and repo controls
- This strategy works well for small and medium scale enterprises. For larger scale enterprises, the idea is that the gitOps management is relegated to each individual team or to a small cluster of related teams. Otherwise, for every N code repos, and products, we need N gitops repos.

GitOps: what a repo looks like

- An average gitops repo is divided into top level directories named after the environments like prod, qa, dev, test, staging ,etc
- There is one template or base directory which houses the template files that are used in the other env directories.
- This is the barebones rest is upto the organization on detail and level of implementation

GitOps: 2 major types

- 2 major types exist
- Left to right - all changes are in a single line
- Shared artifact model

GitOps: Tools

- Terraform - managing infrastructure
- Packer - managing VM creation
- CDK - cloud development kit - use programming languages
- ArgoCD/ FluxCD
- Weaveworks gitOps enterprise suite
- Cloudformation for AWS

Security, Compliance, and drift

- Gitops security is often very clean and precise, since the only real security access needed is at the repository level, which over time is a heavily solved problem with any of the major git vendors like github, bitbucket, gitlab providing excellent access controls.
- When it comes to giving access to infrastructure, the access is generally listed in the repository and a simple drift detection can quickly point out if a particular person has had their access levels elevated with outside resources.
- Drift detection is quite simple, as all gitops tools work with drift detection as a basic building block when applying changes
- With compliance requirements, audit trails become commit histories and process documents become the ci/cd pipelines and readme’s in the repos

Security, Compliance, and drift(contd)

- One of the major pain points with gitOps or anything pull based is that the actual secrets like API keys or ssh keys need to be installed on the servers as well as the CI / CD environments. Which make it hard to manage. A simple solution to this as used by many is to use vaults (like the ones from AWS and hashicorp) to store the secrets. So each entity is only ever configured to read from the vault. This however adds a whole new level of complexity that smaller orgs and individuals might not be ready to handle due to the immense number of moving parts required.
- Another major drawback is that right now, the tools we use for gitOps themselves aren't maintained with the same principles and at times deviate widely from what they preach. As a result, some compliance requirements might entirely reject the automation your organization might have in place. In such scenarios, a fallback to the traditional approaches is often recommended.

Demo

Q&A

Get in touch with me

- Blog: https://blog.codexetreme.com
- Github: @codexetreme
- Twitter: @codexetreme
- Email: grokechrono@gmail.com
- Demo hosted on https://github.com/codexetreme/nitt_cs_2022_gitops_talk

Thanks for tuning in!