Infrastructure as Code (IaC)

Tejas Parikh (t.parikh@northeastern.edu)

CSYE 6225

Northeastern University

Why Infrastructure as Code?

- Virtualization, cloud, containers, server automation, and software-defined networking should simplify IT operations work.
- It should take less time and effort to provision, configure, update, and maintain services.
- Problems should be quickly detected and resolved, and systems should all be consistently configured and up to date.
- IT staff should spend less time on routine drudgery, having time to rapidly make changes and improvements to help their organizations meet the ever-changing needs of the modern world.

Why Infrastructure as Code?

- IT operations teams find that they can’t keep up with their daily workload.
- They don’t have the time to fix longstanding problems with their systems, much less revamp them to make the best use of new tools.
- In fact, cloud and automation often makes things worse.
- The ease of provisioning new infrastructure leads to an ever-growing portfolio of systems, and it takes an ever-increasing amount of time just to keep everything from collapsing.

Why Infrastructure as Code?

- Adopting cloud and automation tools immediately lowers barriers for making changes to infrastructure.
- Managing changes in a way that improves consistency and reliability doesn’t come out of the box with the software. It takes people to think through how they will use the tools and put in place the systems, processes, and habits to use them effectively.
- Change management processes are commonly ignored, bypassed, or overruled by people who need to get things done.
- Organizations that are more successful in enforcing these processes are increasingly seeing themselves outrun by more technically nimble competitors.

What Is Infrastructure as Code?

- Infrastructure as code is an approach to infrastructure automation based on practices from software development.
- It emphasizes consistent, repeatable routines for provisioning and changing systems and their configuration.
- Changes are made to definitions and then rolled out to systems through unattended processes that include thorough validation.

What Is Infrastructure as Code?

- Modern tooling can treat infrastructure as if it were software and data.
- Apply software development tools such as version control systems (VCS), automated testing libraries, and deployment orchestration to manage infrastructure.
- It also opens the door to exploit development practices such as test-driven development (TDD), continuous integration (CI), and continuous delivery (CD).

Goals of Infrastructure as Code

- IT infrastructure supports and enables change, rather than being an obstacle or a constraint.
- Changes to the system are routine, without drama or stress for users or IT staff.
- IT staff spends their time on valuable things that engage their abilities, not on routine, repetitive tasks.
- Users are able to define, provision, and manage the resources they need, without needing IT staff to do it for them.
- Teams are able to easily and quickly recover from failures, rather than assuming failure can be completely prevented.
- Improvements are made continuously, rather than done through expensive and risky “big bang” projects.
- Solutions to problems are proven through implementing, testing, and measuring them, rather than by discussing them in meetings and documents.

Challenges with Dynamic Infrastructure

Server Sprawl

- Cloud and virtualization can make it trivial to provision new servers from a pool of resources. This can lead to the number of servers growing faster than the ability of the team to manage them.
- When this happens, teams struggle to keep servers patched and up to date, leaving systems vulnerable to known exploits.
- When problems are discovered, fixes may not be rolled out to all of the systems that could be affected by them.
- Differences in versions and configurations across servers mean that software and scripts that work on some machines don't work on others.
- This leads to inconsistency across the servers, called *configuration drift*.

Configuration Drift

- Even when servers are initially created and configured consistently, differences can creep in over time:
- Someone makes a fix to one of the Oracle servers to fix a specific user's problem, and now it's different from the other Oracle servers.
- A new version of JIRA needs a newer version of Java, but there's not enough time to test all of the other Java-based applications so that everything can be upgraded.
- Three different people install IIS on three different web servers over the course of a few months, and each person configures it differently.
- One JBoss server gets more traffic than the others and starts struggling, so someone tunes it, and now its configuration is different from the other JBoss servers.
- Being different isn't bad. The heavily loaded JBoss server probably should be tuned differently from ones with lower levels of traffic. But variations should be captured and managed in a way that makes it easy to reproduce and to rebuild servers and services.
- Unmanaged variation between servers leads to *snowflake servers* and *automation fear*.

Snowflake Servers

- A snowflake server is different from any other server on your network. It’s special in ways that can’t be replicated.
- The problem is when the team that owns the server doesn’t understand how and why it’s different, and wouldn’t be able to rebuild it. An operations team should be able to confidently and quickly rebuild any server in their infrastructure. If any server doesn’t meet this requirement, constructing a new, reproducible process that can build a server to take its place should be a leading priority for the team.

Fragile Infrastructure

- A fragile infrastructure is easily disrupted and not easily fixed. This is the snowflake server problem expanded to an entire portfolio of systems.
- The solution is to migrate everything in the infrastructure to a reliable, reproducible infrastructure, one step at a time. The Visible Ops Handbook3 outlines an approach for bringing stability and predictability to a difficult infrastructure.
- DON’T TOUCH THAT SERVER. DON’T POINT AT IT. DON’T EVEN LOOK AT IT.

Principles of Infrastructure as Code

Systems Can Be Easily Reproduced

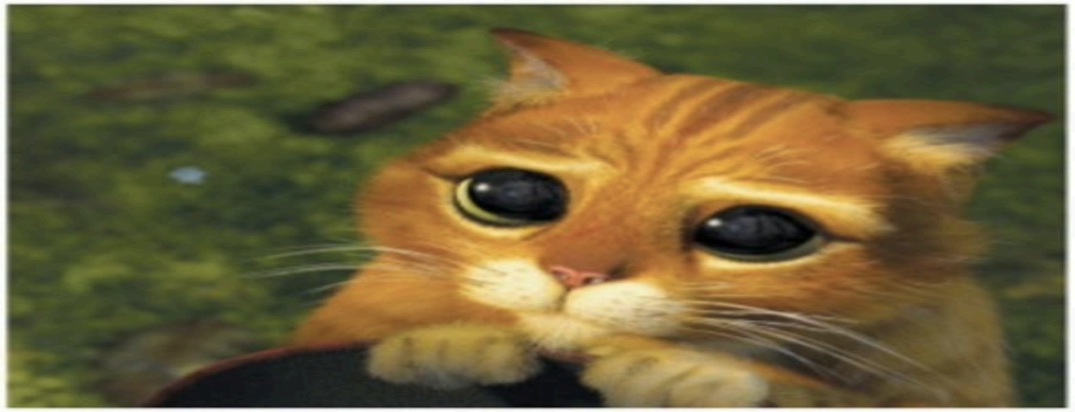
- It should be possible to effortlessly and reliably rebuild any element of an infrastructure. Effortlessly means that there is no need to make any significant decisions about how to rebuild the thing. Decisions about which software and versions to install on a server, how to choose a hostname, and so on should be captured in the scripts and tooling that provision it.
- The ability to effortlessly build and rebuild any part of the infrastructure is powerful. It removes much of the risk, and fear, when making changes. Failures can be handled quickly and with confidence. New services and environments can be provisioned with little effort.
- Approaches for reproducibly provisioning servers and other infrastructure elements are discussed in Part II of this book.

Systems Are Disposable

- One of the benefits of dynamic infrastructure is that resources can be easily created, destroyed, replaced, resized, and moved. In order to take advantage of this, systems should be designed to assume that the infrastructure will always be changing. Software should continue running even when servers disappear, appear, and when they are resized.
- The ability to handle changes gracefully makes it easier to make improvements and fixes to running infrastructure. It also makes services more tolerant to failure. This becomes especially important when sharing large-scale cloud infrastructure, where the reliability of the underlying hardware can’t be guaranteed.

CATTLE, NOT PETS

A popular expression is to “treat your servers like cattle, not pets”



- Pets are given name
- Pets are unique, lovingly raised and cared for
- When they get ill, you nurse them back to health

- Cattles are given numbers
- Cattles are almost identical to other cattle
- Whey they get ill, you get another one

A fundamental difference between the iron age and cloud age is the move from unreliable software, which depends on the hardware to be very reliable, to software that runs reliably on unreliable hardware.

Systems Are Consistent

- Given two infrastructure elements providing a similar service—for example, two application servers in a cluster—the servers should be nearly identical. Their system software and configuration should be the same, except for those bits of configuration that differentiate them, like their IP addresses.
- Letting inconsistencies slip into an infrastructure keeps you from being able to trust your automation. This encourages doing special things for servers that don’t quite match, which leads to unreliable automation.
- Teams that implement the reproducibility principle can easily build multiple identical infrastructure elements.
- Being able to build and rebuild consistent infrastructure helps with configuration drift.

Processes Are Repeatable

- Building on the reproducibility principle, any action you carry out on your infrastructure should be repeatable. This is an obvious benefit of using scripts and configuration management tools rather than making changes manually, but it can be hard to stick to doing things this way.
- Effective infrastructure teams have a strong scripting culture. If a task can be scripted, script it. If a task is hard to script, drill down and see if there's a technique or tool that can help, or whether the problem the task is addressing can be handled in a different way.

Design Is Always Changing

- With iron-age IT, making a change to an existing system is difficult and expensive. So limiting the need to make changes to the system once it's built makes sense. This leads to the need for comprehensive initial designs that take various possible requirements and situations into account.
- Because it's impossible to accurately predict how a system will be used in practice, and how its requirements will change over time, this approach naturally creates overly complex systems. Ironically, this complexity makes it more difficult to change and improve the system, which makes it less likely to cope well in the long run.
- With cloud-age dynamic infrastructure, making a change to an existing system can be easy and cheap. However, this assumes everything is designed to facilitate change. Software and infrastructure must be designed as simply as possible to meet current requirements. Change management must be able to deliver changes safely and quickly.
- The most important measure to ensure that a system can be changed safely and quickly is to make changes frequently. This forces everyone involved to learn good habits for managing changes, to develop efficient, streamlined processes, and to implement tooling that supports doing so.

Practicing Infrastructure as Code

Stack

- A stack is a collection of infrastructure elements that are defined as a unit.
- A stack can be any size. It could be a single server. It could be a pool of servers with their networking and storage. It could be all of the servers and other infrastructure involved in a given application. Or it could be everything in an entire data center. What makes a set of infrastructure elements a stack isn't the size, but whether it's defined and changed as a unit.
- The concept of a stack hasn't been commonly used with manually managed infrastructures. Elements are added organically, and networking boundaries are naturally used to think about infrastructure groupings.
- But automation tools force more explicit groupings of infrastructure elements. It's certainly possible to put everything into one large group. And it's also possible to structure stacks by following network boundaries. But these aren't the only ways to organize stacks.
- Rather than simply replicating patterns and approaches that worked with iron-age static infrastructure, it's worth considering whether there are more effective patterns to use.

Environments

- The term “environment” is typically used when there are multiple stacks that are actually different instances of the same service or set of services.
- An application or service may have “development,” “test,” “preproduction,” and “production” environments.
- The infrastructure for these should generally be the same, with perhaps some variations due to scale.
- Keeping multiple environments configured consistently is a challenge for most IT organizations, and one which infrastructure as code is particularly well suited to address.

VCS for Infrastructure Management

- A version control system (VCS) is a core part of an infrastructure-as-code regime.
- A VCS provides traceability of changes, rollback, correlation of changes to different elements of the infrastructure, visibility, and can be used to automatically trigger actions such as testing.

What to Manage in a VCS

- Put everything in version control that is needed to build and rebuild elements of your infrastructure.
- Ideally, if your entire infrastructure were to disappear other than the contents of version control, you could check everything out and run a few commands to rebuild everything, probably pulling in backup data files as needed.
- Some of the things that may be managed in VCS include:
  - Configuration definitions (cookbooks, manifests, playbooks, etc.)
  - Configuration files and templates
  - Test code
  - CI and CD job definitions
  - Utility scripts
  - Source code for compiled utilities and applications
  - Documentation

What NOT to Manage in a VCS

- Things that might not need to be managed in the VCS include the following:
  - Software artifacts should be stored in a repository (e.g., a Maven repository for Java artifacts, an APT or YUM repository, etc.). These repositories should be backed up or have scripts (in VCS) that can rebuild them.
  - Software and other artifacts that are built from elements already in the VCS don't need to be added to the VCS themselves, if they can be reliably rebuilt from source.
  - Data managed by applications, logfiles, and the like don't belong in VCS. They should be stored, archived, and/or backed up as relevant. Chapter 14 covers this in detail.
  - Passwords and other security secrets should never be stored in a VCS. Tools for managing encrypted keys and secrets in an automated infrastructure should be used instead.

Additional Resources

https://fall2019.csye6225.cloud/