Infrastructure as Code Viewpoint

Delivered by

elyxor

Executive Summary

Businesses must deliver software with higher quality and velocity to their target clientele. One way to enable this is to automate the infrastructure setup and application configuration and deployment. Each stage of the software development lifecycle (SDLC) from local dev to acceptance and eventually to production is performed in an automated, repeatable, and deterministic manner. This process supports the enabling capability of Infrastructure as Code (IaC).

The concept of IaC is that the description of the infrastructure resources and their topology is modeled through code and then managed like software. These descriptor files are managed through version control. Executions are orchestrated via configuration management and Continuous Integration/Continuous Deployment (CI/CD) platforms. This approach cannot be accomplished without collaboration across development, testing, and operations. These constituents define the new approach and implement the IaC framework, tooling, and scripts. IaC processes fall into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

- IaaS: aims to standardize and regulate the compute, storage, and networking components in support of deployed applications. The result will define and create a standardized, base compute desktop or server image with defined networking configurations ready for the platform software to be applied. These images are then provisioned, deployed, and decommissioned on-demand by developers, testers, end users, and infrastructure teams through request or self-service processes. This standardization reduces triage and support costs by the infrastructure team and saves time and provides new capabilities to development and test teams.
- PaaS: layers specified versions of software such as java, web servers, monitoring agents, patches, and other software dependencies on top of the IaaS image to create a Platform. This is the next level of IaC, enabling the image to be application ready.
- SaaS: is layered on top of the provisioned infrastructure and platform and further aims to standardize and regulate the software and configurations required for a given application. Applications can range from the tools to develop and test applications to all the components required to run a bespoke application consisting of a cluster of application servers, message queues, and databases.

At Elyxor, we believe successful software & solution firms will embrace these concepts to realize the following improvements:

- Eliminate Waste - IaC will allow our clients to eliminate manual tasks and increase process velocity.
- Reduce Errors - Reduce errors associated with manual infrastructure, platform, and software provisioning, deployment, and modification.
- Increase Efficiency - Standardized configurations combined with automation enable teams to deliver higher quality at greater business velocity. This provides higher confidence regarding the changes moving through the delivery pipeline for both a bespoke or COTS solution.

These capabilities are critical as organizations take on projects such as data center migrations, migrating to the cloud, creating a self-service portal for developers, automating provisioning and deployments to environments (either Greenfield or Legacy), or adopting new technologies like containers or microservices. Without IaC, organizations will not keep up with the demands of the business to deliver faster with higher productivity and will put themselves at higher risk to quality of service, compliance, or security violations.

Viewpoint and approach

Elyxor believes the following high level approach (Figure 1) will help clarify the major steps needed to introduce IaC to your team(s). Within each step there are details specific to your business where Elyxor can work closely with your team to architect and build the right tech stack detailed implementation plan, and enabling capabilities.

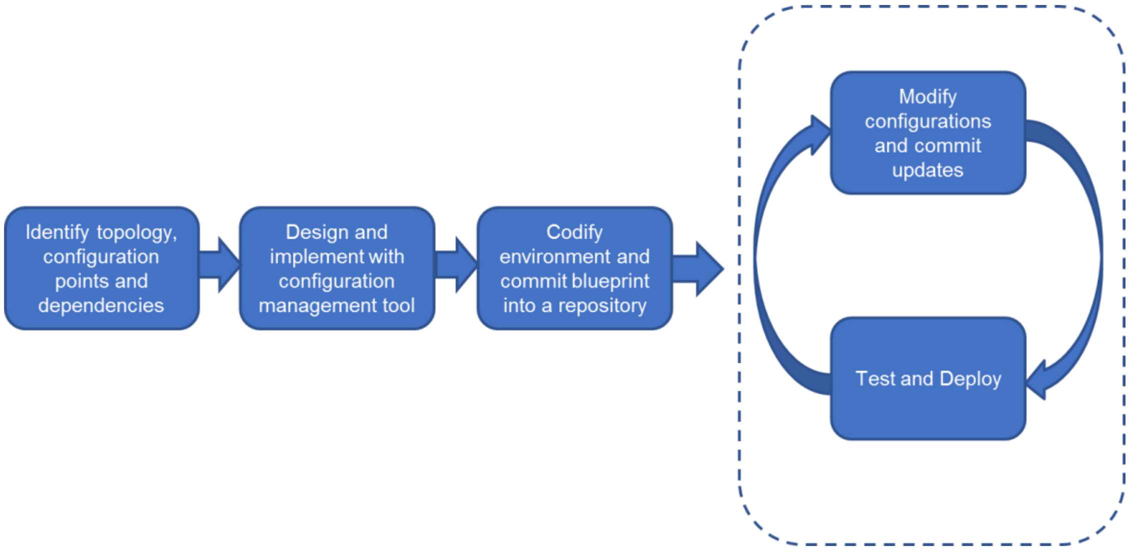
Identify topology, configuration points and dependencies
Design and implement with configuration management tool
Codify environment and commit blueprint into a repository
Modify configurations and commit updates
Test and Deploy

Figure 1: High Level IaC Implementation Approach

The following figure (Figure 2) describes the interaction of the Configuration Management tool with the version control system, and how this is applied to the environment. The result is a topology that reflects the final required infrastructure configuration to support your solution.

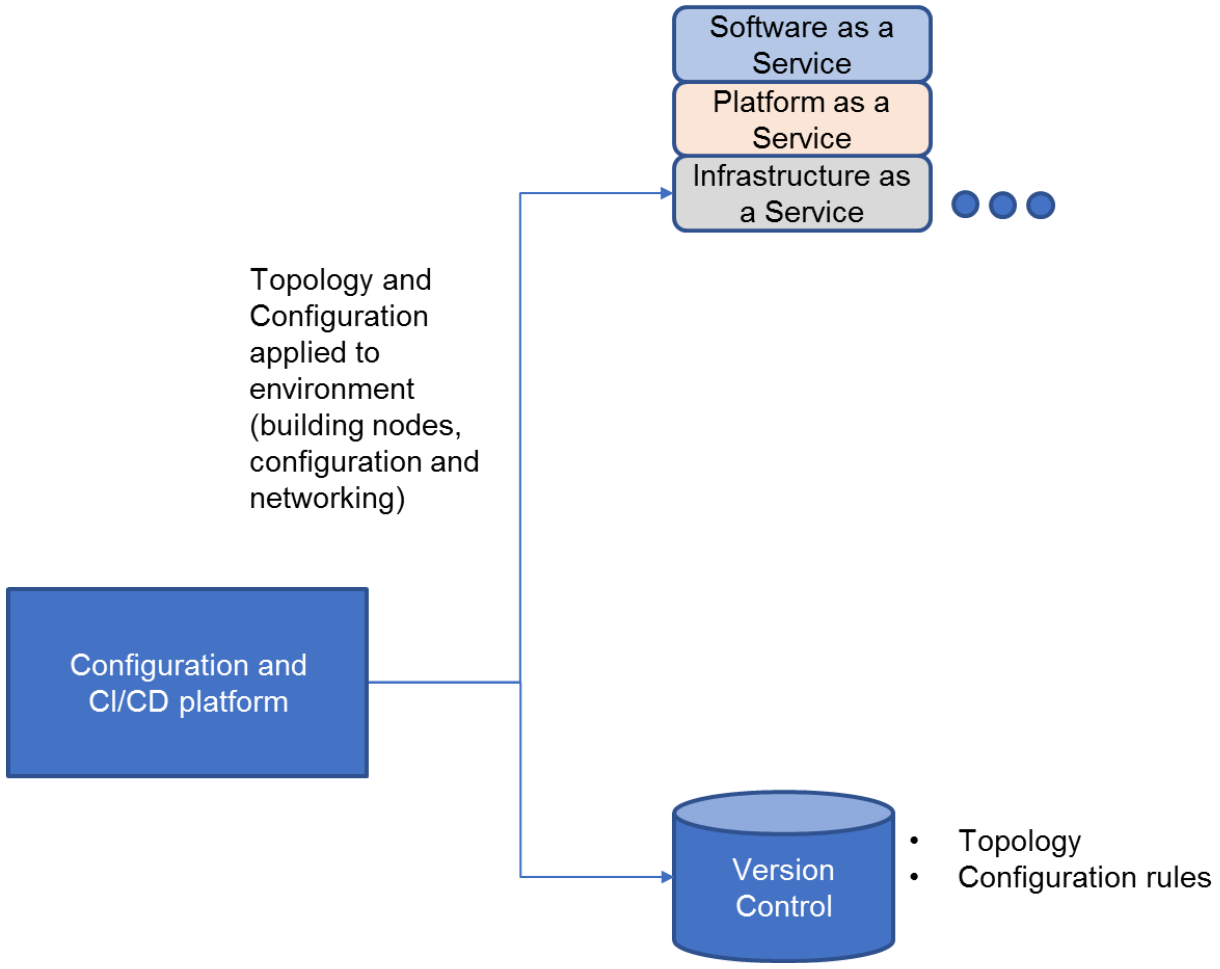
Software as a Service
Platform as a Service
Infrastructure as a Service
Topology and Configuration applied to environment (building nodes, configuration and networking)
Configuration and CI/CD platform
Version Control
Topology
Configuration rules

Figure 2: Conceptual Architecture of IaC Implementation

Various tools are available to support the provisioning, configuration, and CI/CD platform described above. The provisioning tools (i.e. Terraform and Packer) coupled with the config management systems (i.e. Chef, Salt, Ansible, Puppet, CFEngine, or homegrown) automatically provision and deploy the topology to a target environment via a CI/CD framework (i.e. Jenkins or TeamCity), leveraging the topology file that describes the infrastructure for a given platform and system. This topology file is checked into source control and versioned – this is now your self-documented environment blueprint. The resultant CI/CD pipelines enable you to build standard box images to fit your technical need (bare metal, vm, or containers). In addition, as your topology changes, these same pipelines will be able to construct the new topology in automation. You are using code to configure and provision environments. This approach allows you to monitor the deployed configuration by running compliance/drift reports which your business can then schedule the remediation or automatically adjust the infrastructure to remain current.

Benefits

Elyxor has observed that after these improvements, our clients have constructed IaaS + PaaS + application deployments which effectively enable SaaS. The resultant benefits are listed below:

- The team now has a master image that can quickly spin up a new box or apply changes – infrastructure is disposable.
- Elimination of manual adjustment to configuration on multiple servers. This reduces mistakes. Changes are applied via a consistent process allowing for improved delivery times (velocity), auditability, and consistency/repeatability.
- Source code and change control allows for backups, rollbacks, and auditing.
- Reduces risk of one person having all the knowledge (key man failure/risk) or having knowledge distributed like snowflakes in many peoples' heads.
- Better collaboration is enabled across roles.
- A common set of tools and framework enforces standards and improves efficiency – both Development and Operations are using the same SDLC to get features and changes through the environment pipeline to production.
- Developers can now provision hosts that include base operating systems, and are already compliant with security and operations controls.
- This approach also allows for self-service provisioning, deployments and testing.
- Teams now have a process for testing infrastructure code. Start with basic syntax validation and linting, then move on to unit testing, and finally to integration and acceptance testing.
- Mirror production monitoring in the test environment to ensure that infrastructure components don't cause test failures. Note: This does not replace automated testing.

Additional References

- https://techbeacon.com/infrastructure-code-engine-heart-devops
- https://devops.com/configuration-management-vs-application-release-automation/
- https://www.amazon.com/gp/product/1491924357/?tag=kiefcom07-20