elyxor

Boston Office:
1 Harris Street
Unit 7
Newburyport, MA 01950

Little Rock Office:
401 Main Street, Suite 203
North Little Rock, AR 72114

Problem Statement

Organizations need to configure and deploy systems at scale using a reliable, secure, and consistent methodology. Across an organization's business units, systems can vary from simple to complex and need to be supported in production environments by SRE/DevOps teams without suffering the "lowest common denominator" effect in which the process is so generic it is not good for any system deployment. Additionally, SDLC best practices suggest having teams run development and test environments in the exact same manner as prod, increases time to market and operational efficiency; development teams using deployment tools will contribute to their improvement. Consistent tooling eliminates the failure points from the legacy "hand over" process where dev and qa teams use scripts and shortcuts in their dev cycles that are not used by SRE. Many hours are lost to the business when deployments fail due to missing features and configurations that dev environments had, that SRE run production environments do not.

After years of development work, many organizations end up with myriad methods of configuration and deployment, which requires method-specific SRE teams. Furthermore, it becomes difficult for a large organization to apply procedural oversight, ensure security and audit requirements, and enforce production deployment gates consistently across the different methods of deployment.

Most organizations have heterogeneous runtime environments, from highly optimized bare metal servers running in colo to Kubernetes clusters in public/private cloud infrastructure. Matching applications, infrastructure, and network security requirements across the different deployment methods is generally a heavily manual effort which is error-prone especially during high-stress times like production outages.

Finally, these processes make it very difficult to answer the questions: what exact software versions were running; what exact configuration values were used by the software; what data was processed by the system, and in what order did it happen. For companies in regulated industries, answering this question can be somewhere between extremely time consuming and expensive, to nearly impossible.

Elyxor Solution

Operational Simplicity

- Simple and repeatable pattern for configuring and deploying systems at scale
- Environments run in a guaranteed known and deterministic configured state
    - Precise version of each software for each instance
    - Exact configuration values for each instance
- Push-button deploy/rollback mechanism
- Single model for describing systems and their configuration
    - Support simple and complex system footprints
    - Deploy same modelled system into containers, vms, or onto bare metals
- Eventual consistency across environments (i.e. Dev -> QA -> UAT -> Prod) with the ability to see deltas between environments, regardless of infrastructure deployed into
- Support for targeted upgrades that allow individual components to be updated with a smart, efficient deployment mechanism
- Automated application of business and technical gating and validation rules when systems are modified, prior to deployment
- Optimized deployment footprint for systems with global footprints. Especially relevant when moving data is costly due to high-latency or other constraints
- Scheduled or stage deployments into environments at times non-business critical
- Assume zero-trust environments with deployment describing and provisioning allowed inbound and outbound connectivity

Transparency and a single source of truth

- A single model and repository to store system definition, configuration values
- APIs are available for system deployment, configuration, and introspection in real-time
- Visualize a system, running or planned next release, as a tree graph or grid and quickly determine what versions of what apps are running where
- Full audit history of all changes to running systems
- Single place for break glass management
    - Make a change in one place and have it picked up by all instances
    - Auto-commit break glass changes or alert on broken glass
    - Converge to latest committed state
- Easily link running software version with code commits and ticket tracking systems
- Visualize differences between versions of a configured system
    - Release planning: Dry run a to-be-released version of the system against the running version
    - Eventual consistency: See differences between the same system in different environments
    - Auditing: Ability to take any two historically deployed versions and see the difference between on-disk directory and files

Operational Consistency

- Deployments to all environments, regardless of team ownership and infrastructure system is run on, is done using the same tool chain.
- Deployment and monitoring tools have their own SDLC with releases and are deployed with the same tool chain
- Debugging, monitoring, and alerting processes are consistent across environments

Runtime Configuration Model

- TBD: Description of how a system is modelled
- TBD: Describe how devs and SRE interact with it

Initbox Tooling

TBD: Diagram of how initbox takes runtime config and distributes it

Initbox Pattern in Practice

Custom enterprise implementations based on the industry standard jumphost-model were delivered for two premier NY-based financial institutions.

Tech Stack

- Java/Kotlin
- Python
- Ansible
- JSON/YAML
- Git Pipelines
- OpenApi 3