Private clouds do not need to be legacy!

Fabio Alessandro “Fale” Locati

04 February 2024

EMEA Associate Principal Specialist Solutions Architect @ Red Hat

TOC

About me

- GNU/Linux user since 2001
- Working with GNU/Linux since 2004
- Working with Public Clouds since 2009
- Certified AWS and Google Cloud Architect
- Currently working for Red Hat

Why private cloud?

- Technical requirements
- Legal requirements
- Financial requirements
- Organizational decision

What is cloud?

What is cloud?

Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. (Wikipedia) ~~Cloud computing is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user. (Wikipedia)~~
A business model where one party rents to a second party computer system resources, especially data storage (cloud storage) and computing power, with the smallest granularity possible.

- Time: month -> hour -> minute -> second -> millisecond
- Compute: CPU -> Core -> vCore -> fractional vCPU

Lessons we can learn from public clouds

Separation of concerns

- Standardize the interface between infrastructure and workload
- Scalability at workload level
- Workloads have an abstract concept of the physical architecture

Functional business model

- Standardize the interface between infrastructure and workloads
- Bill back infrastructure costs to the workloads owners
- Keep the costs down

Maintain control

- Do not use third-party proprietary software
- Evaluate buy vs build decisions preferring the latter
- Set SLO, measure, iterate
- Be aware of lock-ins
  Product between the **probability** that a component will require substitution during the solution life and the **total costs** in case of substitution.

Technologies considerations and bets

KISS

- Reduce the complexity of your system to a minimum
- Prefer build-time complexity over run-time complexity
- Minimize the amount of services available

Containers

- Use a Kubernetes distribution
  - DIY/Community
  - Commercial
    - Fully open source
    - Trustworthy company
    - "Valuable" offering

Automation

- Use an immutable approach to infrastructure
- Version the infrastructure (eg: gitops)
- Automate process end-to-end

Conclusions

Putting all together

- Infrastructure
- API
- Workloads

Putting it all together - Infrastructure

- Create/Architect for multiple DataCenters (and multiple clusters) but hide them from the workload developer
- Deploy Kubernetes container platform clusters on bare-metal
- Use a tool to manage and abstract the clusters (eg: Open Cluster Management)
- Set SLO, measure, iterate
- Automate all the infrastracture pieces and configuration

Putting it all together - API

- Define discrete “regions” based on non-technical requirements, like legal frameworks (eg: eu, us)
- Standardize the Kubernetes APIs as the only interfaces between infrastructure and workload
- Start providing only: OCI registry, Object Storage, and a very limited subset of Kubernetes objects (eg: Pods, Deployments, Stateful SetsServices, PV, PVC, ConfigMaps, Secrets)
- Provide more services once you have a good strategy to support them and many of your users are already using the technology (eg: Databases)

Putting it all together - Workloads

- Create a simple UX to submit the creation/update/deletion of workloads objects
- Store workloads objects in a versioned storage (eg: git) and automate deployment
- Require (opt-out?) applications resilient to restarts, replications, etc.

Thanks

mail@fale.io