Chaos Engineering and Service Ownership at Enterprise Scale

Jay Hankins (he/him)
Lead Chaos Engineer at Salesforce

Back to 2015

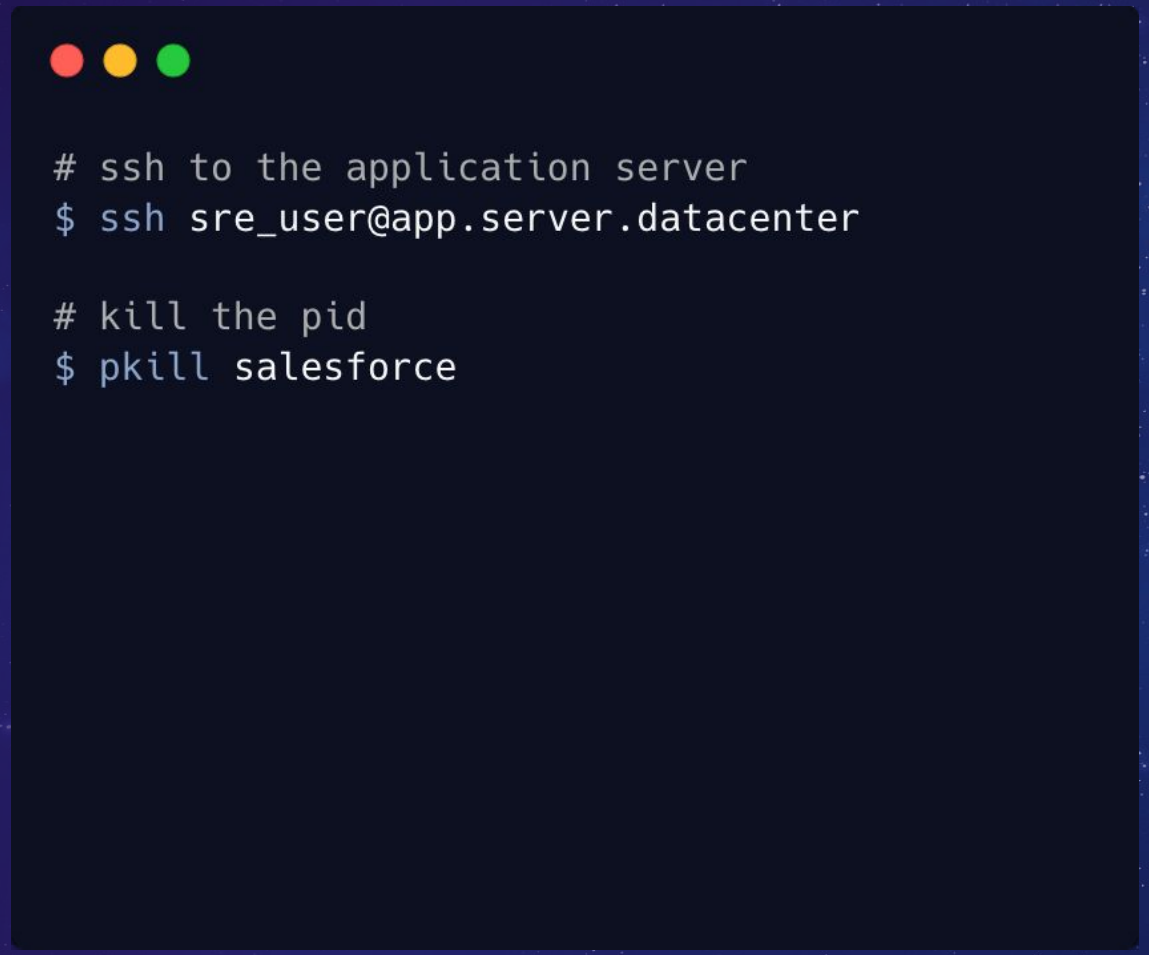
Salesforce infrastructure in 2015:

- first-party data centers
  - inelastic infrastructure
- simpler application footprint, security controls, and ownership models
- SREs owned the availability of “everything”
  - and had widely-scoped privileged shell access

Salesforce Chaos Engineering in 2015:

- Relied on privileged shell access:
  - Killing processes
  - Rebooting hosts
- Tight partnership with network or data center engineers:
  - Turning off ports on network switches
  - Data center cold restarts

A Typical Game Day Exercise in 2015



A typical chaos game day in this era:

- An SRE would use privileged shell access to run commands manually.
- They would then observe critical host and application metrics, while the game day project team would scribe findings.

Evolving Infrastructure and Service Ownership

- Business needs demanded larger and more flexible infrastructure
  - Sales growth, new products, companies acquired, new regulations, etc.
- Public cloud (Hyperforce) infrastructure enforces new internal requirements and operational practices.
  - New infrastructure brings a bevy of “foundational” services such as PKI, secrets, ingress and egress proxies, etc.
  - It also eliminates most interactive (shell) access.
- Salesforce fully embraces service ownership.
  - No more “throwing it over the fence” to SRE.

Chaos Engineering: a part of Service Ownership

Challenges:

- In a service ownership world, SRE has less of a centralized role
- A centralized game day team can't learn all the architectures and edge cases of new/designed services for public cloud
- New technical constraints around privileged (shell) access made previous chaos approaches unfit for Hyperforce.

Shifting our approach:

- Service owners know their service better than anyone else.
- Shifting left in the development cycle reduces turnaround time on discovering and fixing issues.
- We should deliver a Chaos Engineering Platform that lets service owners run chaos experiments safely and easily.

Major Scale and Shift-Left Challenges

1. Size and shape of our AWS footprint
2. Granularly attacking multi-tenant compute clusters
3. Simplifying discovered inventory and access
4. Maintaining safety, observability, and outcomes

Challenge 1: Our AWS Footprint

Challenges:

- Our Core CRM product is hundreds of services spanning 78 AWS accounts.
- Services may have their application, database, cache, etc. in separate accounts.
- It's infeasible for humans to log into every account to inject failures.

Requirements:

- We need a privileged chaos engineering platform that can run attacks in AWS in multiple accounts simultaneously.

Challenge 2: Multi-tenant Kubernetes Clusters

Challenges:

- Services are deployed across many namespaces × clusters.
- Service owners should only be able to attack their service, not shared services or the cluster itself.
- Service owners may know less about Kubernetes infrastructure.

Requirements:

- We need a privileged chaos engineering platform that can orchestrate attacks in multiple namespaces and clusters simultaneously.
- We need the platform to provide failures without requiring ad-hoc cluster configuration, service accounts, etc.
    - Service owners should only need minimal knowledge of the k8s API and not need to deploy chaos workflows, configmaps, etc.

Challenge 3: Inventory and Role-Based Access

Challenges:

- Discovering and accounting for all the different resource types owned by a service team
  - e.g., a K8s deployment, an S3 bucket, an RDS database,
- Enforcing RBAC and controlling blast radius based on job role and service ownership

Requirements:

- Our chaos platform should integrate with, discover, and group all sorts of infrastructure resources.
- Our chaos platform should integrate with SSO to match service owners to their services
- Our chaos platform should make use of opinionated tagging/labeling to match group services and service owners

Challenge 4: Safety, Observability, and Outcomes

Challenges:

- What if there is an ongoing incident or maintenance? It might be unsafe for service owners to run experiments.
- How should service owners measure the success of their chaos experiments, and how do we track improvement?

Requirements:

- Our chaos platform should integrate with our change and incident management database and refuse to attack when it's unsafe.
- Service owners should measure their chaos experiments through the same SLOs and monitors that are used in production.

Recommendations for a self-service chaos platform

1. Chaos tooling should be multi-substrate to support future flexibility.
2. Make use of RBAC and tags, labels, etc. to control blast radius and limit attack access.
3. Prioritize extensibility to integrate with custom systems, like we did for change management.
4. Seek out a sophisticated toolbox of attacks to support both large-scale GDE-style experiments AND precision attacks that affect individual services/teams.
5. Use SLOs, make them part of your hypotheses, and make sure service owners observe experiments as they would observe production.

The Ongoing Role of Game Day Exercises

Optimize for purpose and expertise.

1. Service owners take charge of concrete technical fixes
2. GDE teams can support:
    a. compliance exercises, such as SOC2, data RPO/RTO
    b. Organizational/people & process chaos, including incident response
    c. Shared IT service chaos, such as attacking your wiki/operational runbooks
    d. Table-top exercises to help service owners scope their attacks

Thank you