Designing for Operations

A presentation by:
Rutwik Ghag
Puneet Shetty

Some context

- This chapter explores how to design distributed computing systems
- DC system: Many small parts make one machine
- Multiple servers work together to accommodate large volume of requests
- Large scale machine, many components doing specialized tasks
- Hardware failure is an expected part of the system, everything else works around it
- Systems evolve over time, hence components are loosely coupled, no strong dependencies

Some more context

- This chapter explores how to design distributed computing systems (a server) for common operational tasks running a service/software
- It also describes how to rebuild a non-operations-focused architecture.
- Designing for operations entails ensuring all typical operations work in addition to the primary function
    - Maintenance, updates, and monitoring are normal operational functions. Early planning must consider these difficulties.
- A minor percentage of a service's life cycle is spent creating features for clients.
- The service's operation takes up most of its life.

Operational Requirements

- Software is built based on end-user’s needs
- Rarely is the needed functionality considered in the design phase
- Systems administrators are thus lacking key interaction points
- Typically, devs and managers leave debugging to the operations team
- When the operations team designs features for a distributed computing system, there are some features that are essential to the functioning of the system but do not affect the functioning of the application or service
- These operational features are as follows:

#1 Configuration

- The system must be configurable by automated means, including initial configuration, and be able to do the following:
  - Make a backup of the settings and store it
  - View the differences between multiple archived copies
  - Archive the current settings without taking the system down

#2 Startup and Shutdown

- The service should startup automatically when the system boots up
- The machine should include the proper OS to shut the service down properly
- If the machine crashes, the next restart must perform data validation/repairs
- The time required to startup/shutdown should be documented for DRP

#3 Queue Draining

- There must be an orderly shutdown process that can be used to take the system out of service for maintenance
- A drain occurs when the service is told to stop accepting new requests but complete any requests that are currently under operation. This is called lame-duck mode
- This is important when using a load balancer (one front end and multiple back ends), because upgrades are done one replica at a time

#4 Software Upgrades

- It must be possible for software upgrades to be implemented without taking down the service.
- Software is located behind a load balancer and upgraded by swapping out replicas as they are upgraded.
- Some systems can be upgraded while online, but that is risky, requires careful design and extensive testing
- Non-replicated systems are difficult to upgrade without downtime
    - The general system is to make a clone, upgrade the clone and then swap it
    - Difficult because clone needs to be made when no transactions were in progress

#5 Backups and Restores

- Must be possible to backup and restore the service’s data while the system is up and running
- Live backups are usually done by backing up the read-only replica of the database
    - It is common to have a replica dedicated for backups
- The system should be able to restore a single account without having to restore all accounts

#6 Redundancy

- Many reliability and scaling techniques are predicated on the ability to run multiple, redundant replicas of a service, hence services should support these configurations
- Services must be designed to work behind a load balancer for large scale applications, if not there can be issues
    - Eg: Users login credentials will need to be stored on replicas as well, not just the local server, to avoid being asked to login repeatedly in the case that the load balancer redirects the request to a different server

#7 Replicated Databases

- Systems that access databases should do so in a way that supports database scaling
- The most common way to scale is to create one or more read-only replicas.
- The master database does all transactions that mutate the database. Updates are then passed to the read-only replicas via bulk transfers.
- Most database access is read-only, so the majority of work is off-loaded to the replicas.
- The replicas offer fast, though slightly out of date, access. The master offers full-service access to the “freshest” data, though it might be slower

#8 Hot Swaps

- Service components should be able to be swapped in or out of their service roles without triggering an overall service outage.
- Software components may be self-sufficient for completing a hot swap or may simply be compatible with a load balancer or other redirection service that controls the process.
- Hot-swappable devices can be changed without affecting the rest of the machine.
  - For example, power supplies can be replaced without stopping operations
- Hot-pluggable devices can be installed or removed while the machine is running. Administrative tasks may be required before or after this operation is performed.
  - For example, a hard drive may not be recognized unless the operating system is told to scan for new disks, eg: USB, new Graphic cards

KAHOOT

Choose mode - Kahoot!

#9 Toggles for Individual Features

- A configuration setting (a toggle) should be present to enable or disable each new feature.
- This allows roll-outs of new software releases to be independent of when the new feature is made available to users
  - Eg: Instead of pushing out a new update, it can be a part of the software but only be enabled later at a desired time. This is a flag flip
  - It is easier to disable the individual feature via a flag flip than to roll back to the previous version

#10 Graceful Degradation

- Software acts differently when it is becoming overloaded or when systems it depends on are down.
- The operation is based on the idea that “Something is better than nothing”
  - For example, on a web site, normally users receive the rich interface. However, if the system is overloaded or at risk of hitting bandwidth limits, it switches to the lightweight mode.
- A service may become read-only if the database stops accepting writes (a common administrative defense when corruption is detected)
- If a database becomes completely inaccessible, the software works from its cache so that users see partial results rather than an error message

#11 Access Controls and Rate Limits

- If a service provides an API, that API should include an Access Control List (ACL) mechanism, and also determines rate-limiting settings
- An ACL is a list of users, along with an indication of whether they are authorized to access the system.
- Types of ACL:
  - A list of users that are permitted access; everyone else is banned. This is a default closed policy; the list is called the whitelist.
  - The reverse would be a default open policy; the default is to give access to all users unless they appear on a blacklist
  - An ordered list of users/groups annotated as either being “permitted” or “denied.” If a user is not mentioned in the ACL, the default action might be to permit the user (fail open) or, alternatively, to deny the user (fail closed)
- ACLs can indicate rate limits: different users might be permitted different queries per second (QPS) rates, with requests that go over that rate being denied.
  - Eg: Pay a premium to get faster speeds while using a VPN

#12 Data Import Controls

- If a service periodically imports data, mechanisms should be established that permit operations staff to control which data is accepted, rejected, or replaced.
- If a bad record causes a problem with the system, one must be able to block it via configuration rather than waiting for a software update.
- Such a system uses the same whitelist/blacklist terminology we saw earlier
- We also need a way to augment an imported data source with locally provided data, using an augmentation file of data to import.
- Establishing a change limit can also prevent problems by requiring approval for operations that change a huge amount of data

#13 Monitoring

- Each component of a system must expose metrics to the monitoring system
- These metrics are used to monitor availability and performance, for capacity planning, and as part of troubleshooting

#14 Auditing

- Logging, permissions, and role accounts are set-up to enable the service to be examined for, and pass, security and compliance audits
- Data quality needs to be assessed frequently to ensure compliance with regulations
- When companies use the public cloud, their data needs to be in accordance with the local law

#15 Debug Instrumentation

- Software needs to generate logs that are useful when debugging. Such logs should be both human-readable and machine-parseable.
- A debug log usually records the parameters sent to and returned from any important function call
- It helps understand errors and fix them to keep the system running
- Every developer should add a debug logging statement for any information seen fit.
- The documentation on how to consume such information is the source code itself, which should be available to operations personnel

#16 Exception Collection

- When software generates an exception, it should be collected centrally for analysis.
- A software exception is an error so severe that the program intentionally exits.
- Certain data corruption scenarios are better handled by a human than by the software itself.
- It is common to use a software framework that detects exceptions, gathers the error message and other information, and submits it to a centralized database. Such a framework is referred to as an exception collector

#17 Documentation for Operations

- Developers and operational staff should work together to create a playbook of operating procedures for the service.
- It is critical that every procedure include a test suite that verifies success or failure.
- Documentation is a stepping stone to automation. Processes may change frequently when they are new. As they solidify, you can identify good candidates for automation

Implementing Design for Operations

- Features designed for operations need to be implemented by someone
- Every software has features, but these need to be functional in the software
- There are 4 main ways to add features to the software:
    - Build features in from the beginning
        - It is extremely rare that you will encounter this scenario outside of large, experienced shops like Google, Facebook, or Yahoo
    - Request features as they are identified
        - If you have access to the developers, features can be requested over time as you identify the need for them through daily use
    - Write the features yourself
        - When developers are unwilling to add operational features, one option is to write the features yourself.
    - Work with a third party vendor
        - The vendor acts as your development team to accommodate your needs

Improving the Model

- Good design for operations makes operations easy. Great design for operations helps eliminate some operational duties entirely
- It is equivalent to hiring an extra person
- When possible, strive to create systems that embed knowledge or capability into the process, replacing the need for operational intervention.
- Great design changes the job of the operations staff from performing repetitive operational tasks to building, maintaining, and improving the automation that handles those tasks.
- A common operational task is future capacity planning—that is, predicting how many resources will be needed 3 to 12 months out.

Summary

- Services should include features that benefit operations, not just the end users
- Operations staff need many features to support day-to-day operations
- Services should degrade gracefully when there are problems, rather than become completely unusable.
- Systems are easier to maintain when this functionality is designed into them from inception. Operations staff can work with developers to ensure that operational features are included in a system, particularly if the software is developed in-house

Thank You!