DevOps for Defense

September 2020

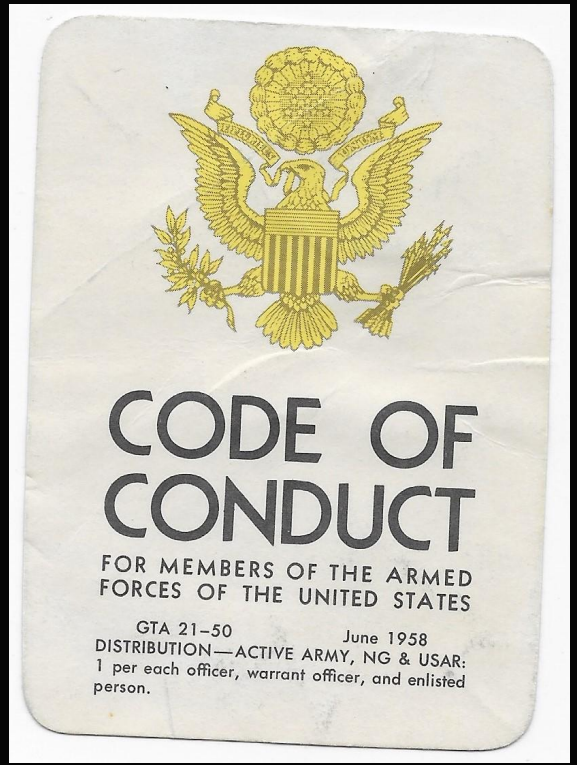
Scaling Services in the Cloud

Vadim Uchitel

https://devopsfordefense.org
https://www.meetup.com/DevOps-for-Defense/
https://github.com/jondavid-black/DevOpsForDefense
devopsfordefense@gmail.com
https://twitter.com/devops4defense

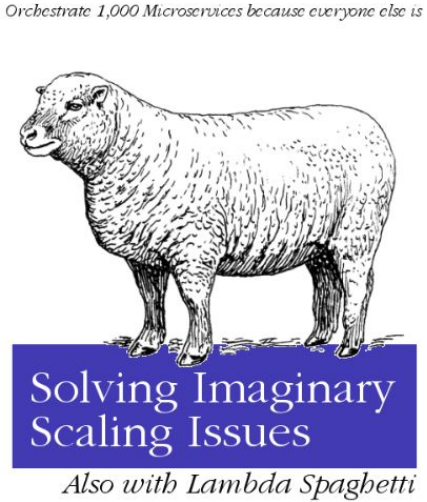
DevOps for Defense Meetup: Code of Conduct

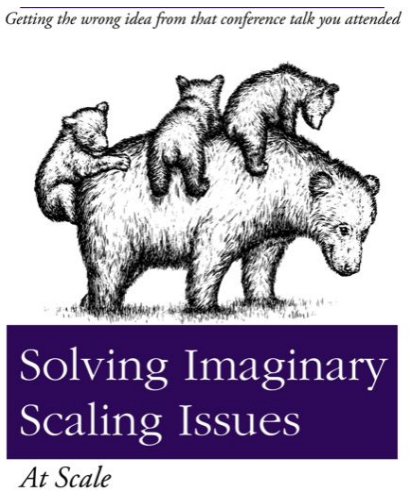
- UNCLASSIFIED ONLY!!!!
- Treat each other with respect and professionalism.
- Do not talk about private, sensitive, or proprietary work.
- Do talk about your experiences, needs, desires to improve work in our domain.
- Do share your thoughts.
- Do learn from others.
- Do mute yourself while others are speaking!
-



Scaling services in the cloud

by Vadim Uchitel





About presenter (https://www.linkedin.com/in/vadim-uchitel-2037b91/)

- Fetch Rewards (07/2020 - now)

  -> Millions of users
- Shipt (<3 years)

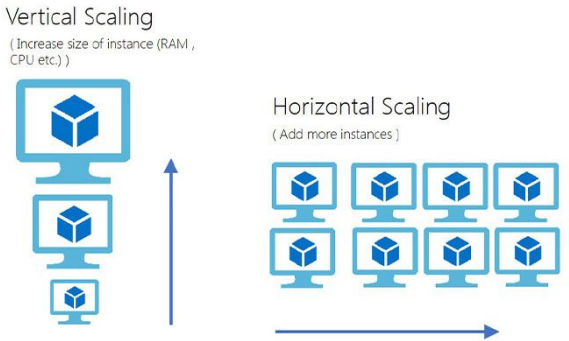
  -> Hundreds of thousands users
- SAIC / Leidos (11 years)

  -> Hundreds of users
- CFDRC / ESI-CFD (10 years)

  -> Hundreds of users

What is scalability?

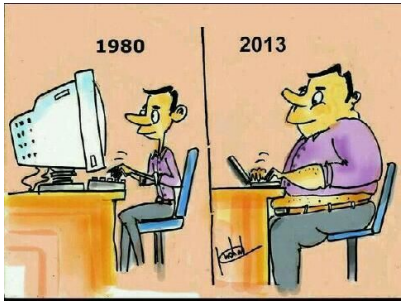
Scalability is the ability of a process, network, or software to grow and manage increased demand or complexity



- Scale Vertically - Scale Up
  - Moving to a bigger virtual machine
  - Adding more CPU / memory
- Scale Horizontally - Scale out
  - Adding instances
- Diagonal scaling
  - Combination of vertical and horizontal scaling. Scale vertically first until you reach a preset limit and then scale the system horizontally.
- Global Scaling
  - Scale across regions

Moore’s law

- 1965 forecast that the number of components on an integrated circuit would double every year
- (revised in 1975) doubling of transistors on a chip every 18-24 months
- Since 2010, it has been 2.5 years and the rate continuous to slow down
- More importantly - single core performance (clock speed) improvements have been nearly flat the last few years due to hitting heat dissipation issues
- Recent CPU improvements were in increasing number of cores (horizontal scaling)

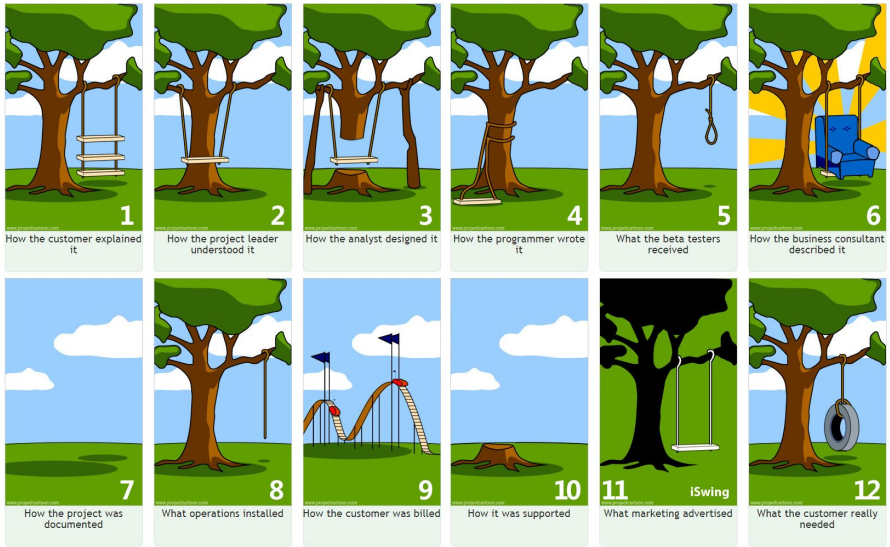


Does cloud computing provides unlimited scalability?

- Horizontal scaling -> Yes (from the consumer point of view)
- Global scaling -> limited to number of supported regions
- Vertical scaling -> No

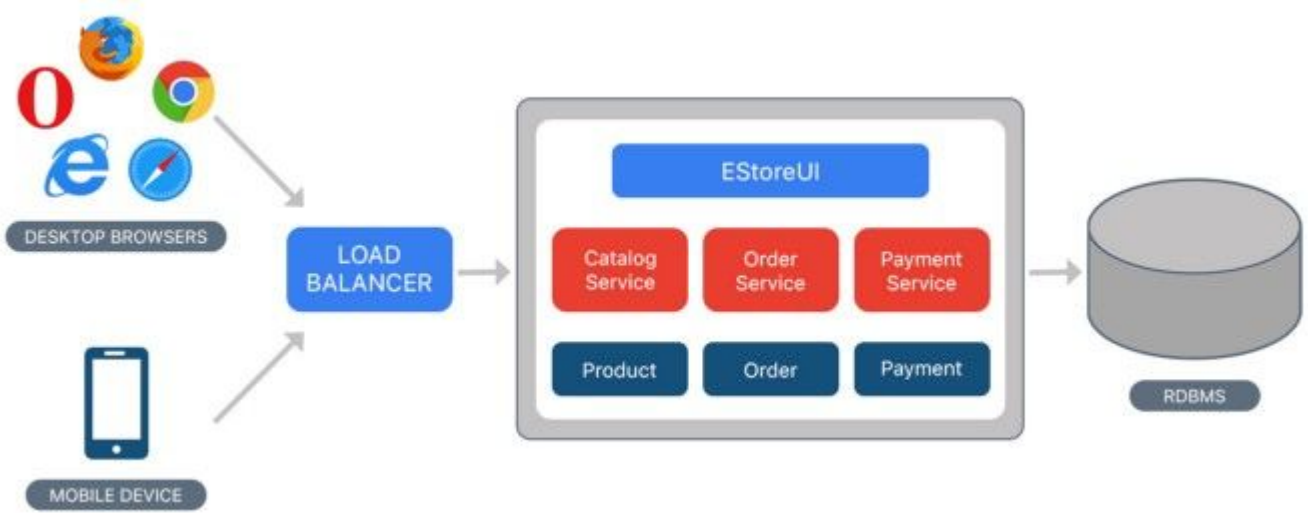
If cloud providers could provide unlimited vertical scaling then this presentation would have only 1 slide - “**use vertical scaling to run your code**”.
- However, CPU / memory / GPU / etc. resources have hard limits; thus, the only way to truly scale services in the cloud is to use horizontal scaling.

Prior to scaling, let’s talk about building web services



Always start with a monolith

- Every project / startup typically starts with a single, monolithic application that is connected to a single datastore



Are monoliths really that bad?

Pros:

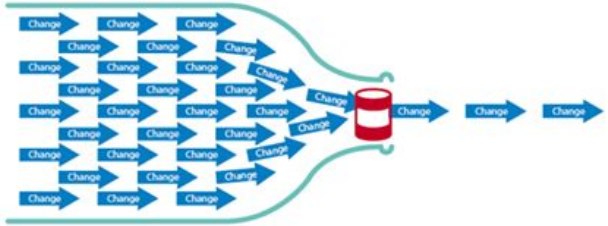
- Very easy to start, do POC and start to grow your business
- Easy to add new developers at the beginning
- Easy to deploy
- Easy to exchange data between modules

Cons:

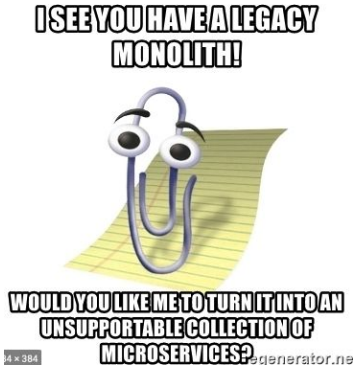
- Parts cannot be scaled independently
- Maintenance becomes nightmare at scale
- Single point of failure

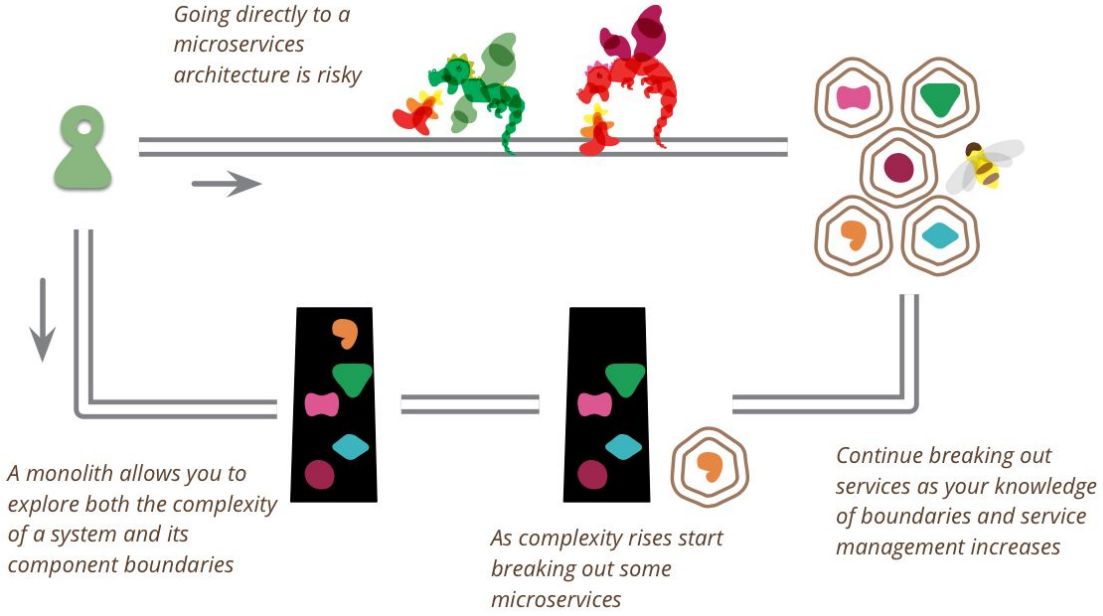
Database becomes bottleneck

- RDBMS (typically) runs on a single server; thus bounded by vertical scaling issues.
- Major issue with RDBMS is the maximum number of available connections. For example, PostgreSQL for each connection (it is a TCP connection) creates a new process, which consumes CPU, memory and file descriptor resources. Thus, sooner or later, growing company will overgrow its database.
- Even the biggest AWS RDBMS instance cannot handle the increased workload

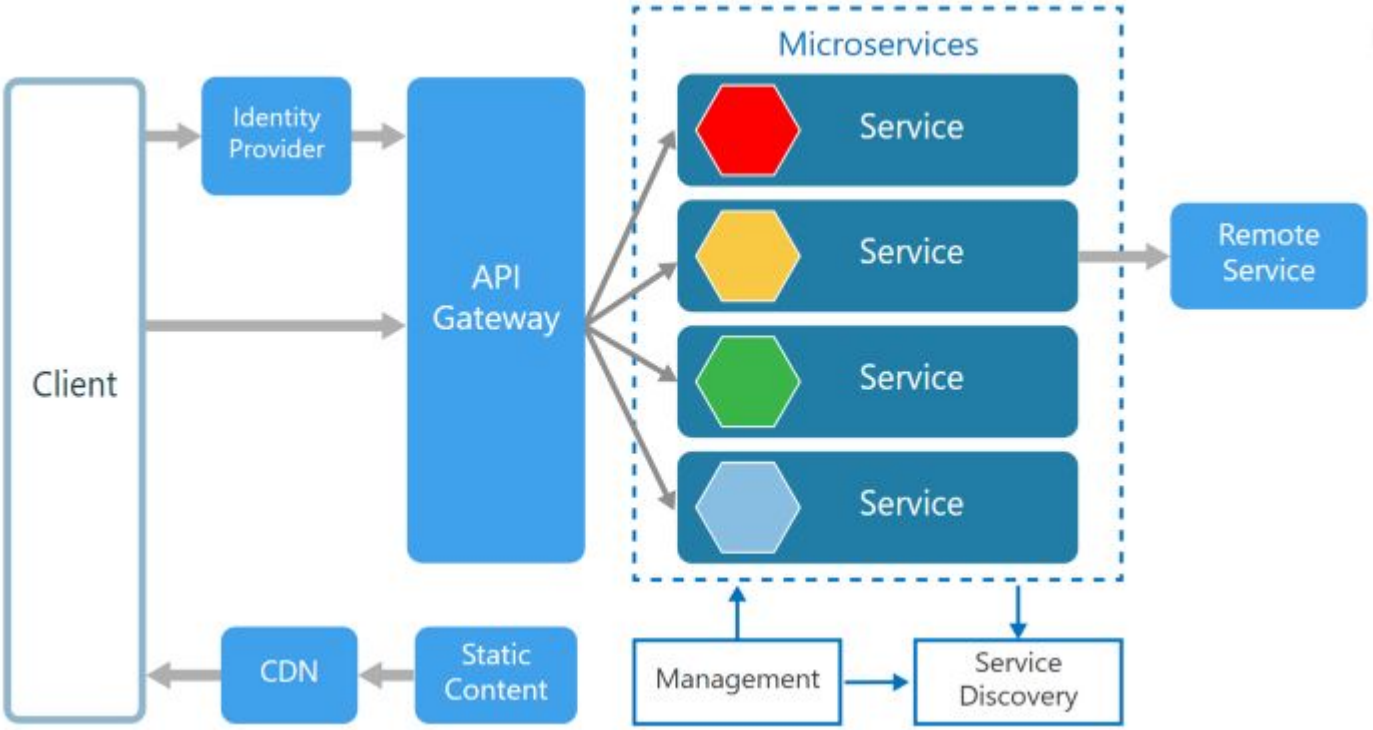


Transition to microservices



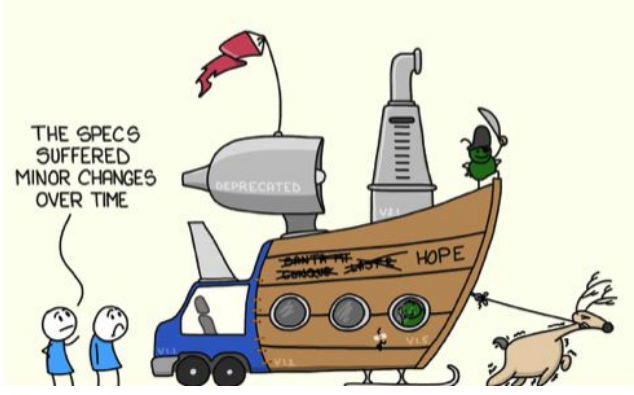


Microservices pattern emerges



Major reasons for moving to Microservices

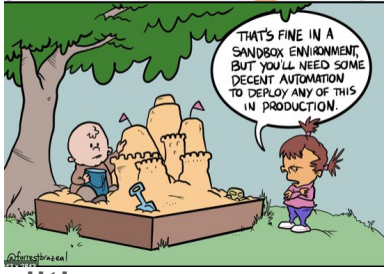
- Remove datastore bottleneck
  - Each microservice has its own datastore; thus, removing the major system bottleneck
- Solve Conway Law problem
  - “Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure.”
  - Monolith provides a perfect solution for a small organization
  - As company continues to grow, more teams need to be added with responsibilities for specific domains, i.e. Microservices
- Cleanup legacy code



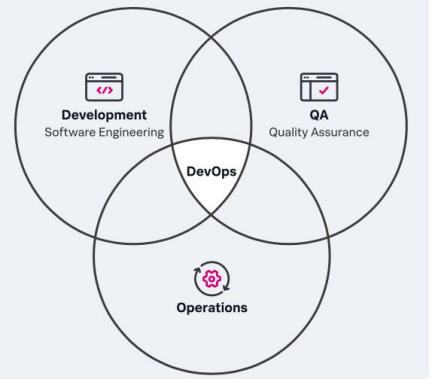
Known challenges with Microservices

- Issues with network reliability and latencies
- Data duplication
- Orchestration
- Integration
- Observability
- Need for API Gateway
  - Unified point of entry that hides backend implementation
  - Front end only calls API Gateway, backend devs are responsible for proper routing
  - Provides easy ability to migrate routes between services
  - Handling authentication and security
  - BFF (backend for frontend), i.e. ability to combine multiple backend calls into single response
  - Another piece of architecture to maintain

DevOps Culture



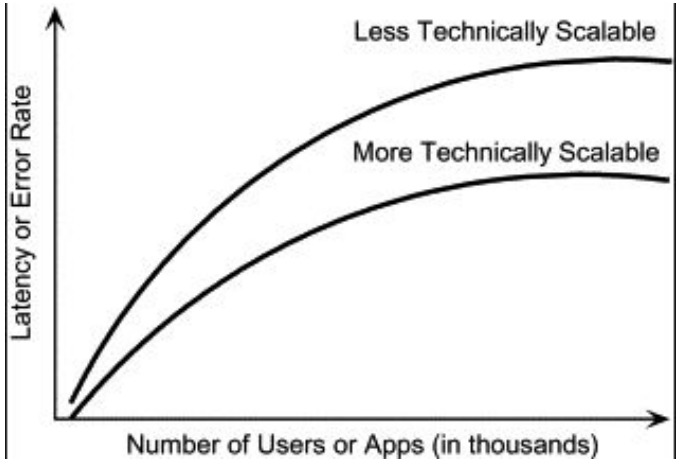
- Running microservices at scale is harder than dealing with monolith
  - Multiple deploy pipelines
  - Container/VM orchestration
  - Integration tests
  - Infrastructure monitoring
  - Managing database instances
  - Managing infrastructure costs
    - Developers have tendency to create cloud resources and forget about them; thus, causing $$$$ losses
  - Managing logging, tracing, telemetry solutions

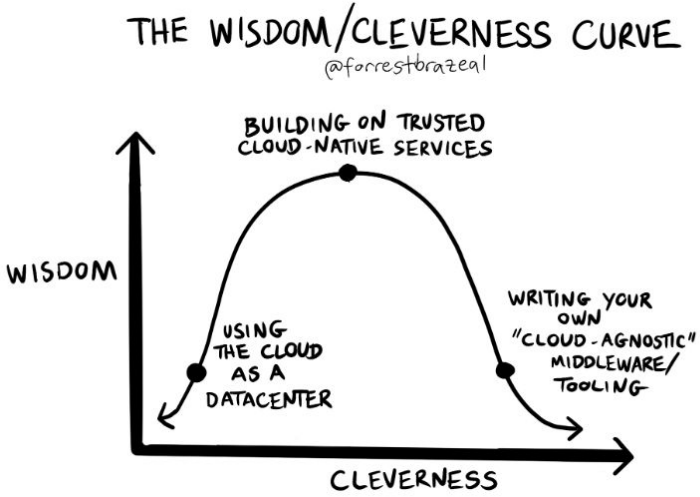


Importance of robust CI/CD

- Developers should have full confidence in CI/CD pipeline in order to deploy constant changes to production
- Some of the features of robust CI/CD
  - Performance. Faster CI/CD -> more often developers deploy code -> more features
  - Fully automated. All human interventions should be optional, not required
  - Automation over documentation. If you need to document your CI/CD pipeline then you don’t have a robust CI/CD pipeline
  - Built-in automatic notification on successes and failures
  - Automatic notification of what is being deployed
  - Rollback

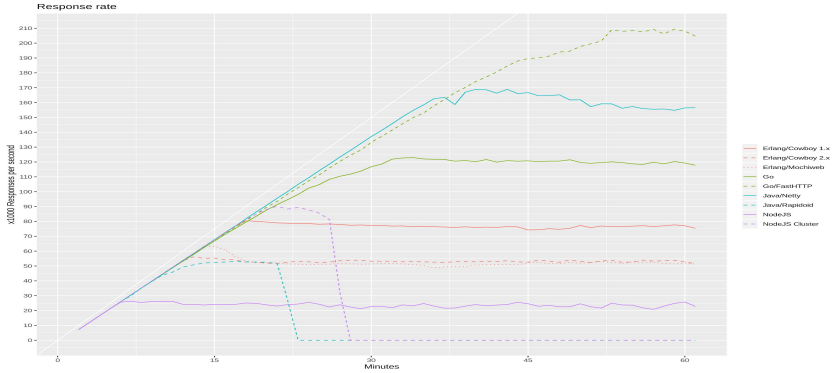
Horizontal scaling while minimizing costs



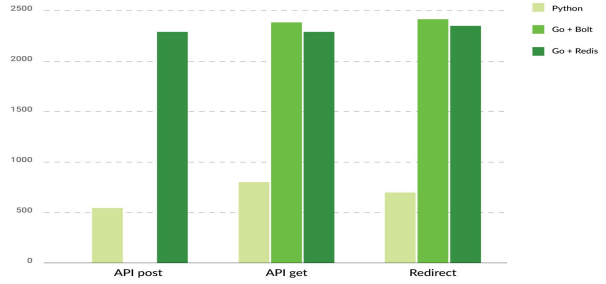


Language choice and its effect on performance

- Using high performance language, such as Golang, allows to have less instances and smaller containers & virtual machines.
- At the graph to the right, Go's FastHttp router can handle 210k RPS vs Java's netty 170k RPS. That is ~20% extra h/w that you don't need to have.
- Comparing to Ruby on Rails, and Python with Django, Golang provides 2-3x performance improvement.
- Note: Rust is gaining popularity in certain use cases (especially WebAssembly), as it provides better performance



https://stressgrid.com/blog/webserver_benchmark



https://djangostars.com/blog/my-story-with-golang/

Miracle of Redis (in memory key-value datastore)

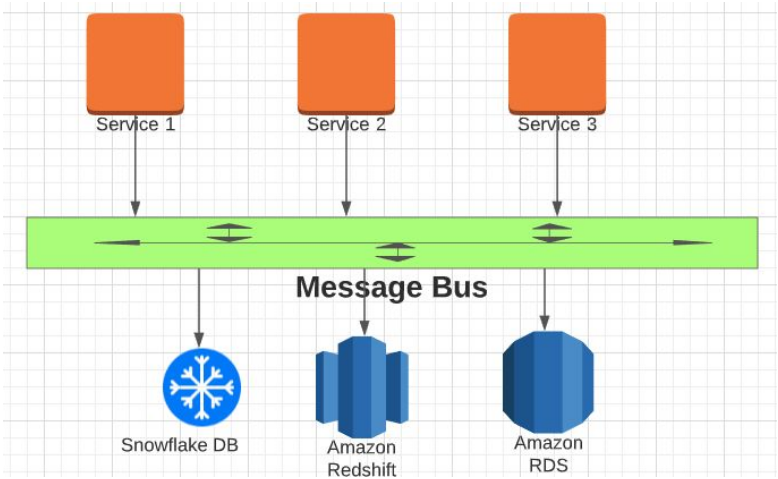
- All Redis data resides in the server’s main memory
  - No round trip to disk, that is a staple of traditional datastores
- Supports an order of magnitude more operations and faster response times. The result is blazing fast performance with average read or write operations taking less than a millisecond and support for millions of operations per second.
- Vast variety of data structures to meet any application needs.
- Primary-replica architecture with asynchronous replication where data can be replicated to multiple replica servers
- Streams and Pub/Sub

Minimize database access in user workflow

- Reducing response time leads to several things:
  - Happy customers
  - Less h/w to use
  - Less $$$ to spend
- Reduces response time on user request and shields the primary datastore from increased workload
  - Achieves the goal to handle increased payload on demand
- Multiple ways to achieve fast response:
  - Use Redis as a cache in front of main database
  - Use Redis as a primary datastore
  - Use RAM, it is cheap
  - Use asynchronous updates of the primary datastore
  - Make your primary datastore eventually consistent

Async update of the primary datastore & Data Lake

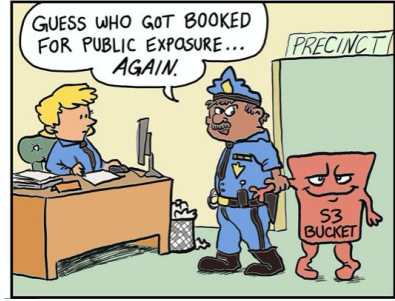
- Data Lake (storage of raw data that can be later used for analytics, recovery, etc.) needs to be kept out of operational workflow and updated via asynchronous streams
- Message/Event Bus architecture that can be implemented using SNS / Kinesis / Kafka / MQs / etc. is a great way to separate analytics and long term storage from real time operations



Load Testing

- **Load testing** is performed to determine a system's behavior under both normal and anticipated peak **load** conditions
- Locust https://github.com/locustio/locust is a great tool to perform load testing that can test HTTP, WebSockets, etc.
- Testing 10x expected production workload is recommended
- Load testing is extremely important to size containers and VMs and need to be performed continuously, including existing production services
- Running load tests allow on-call engineers to sleep better on weekends when system load is at its peak

Don’t forget about Sec part in DevSecOps



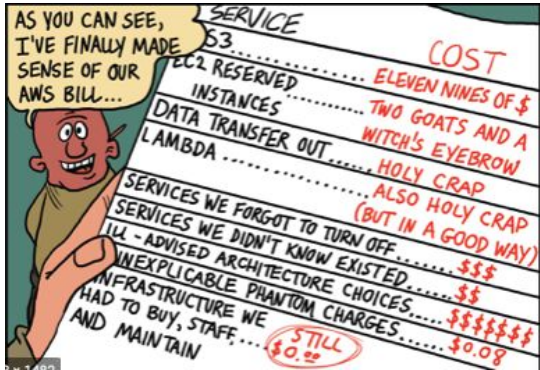
- Security exposure can ruin any company
- Scaling and growing is never an excuse for bad security practices
- Bad actors are out there and they want to steal your data and your money
- Every engineer has to think about security. Security can not be achieved by bureaucratic rules, but only by engineers always thinking and practicing it.
- Best security practices:
  - Microservices need to be in private VPC without public access
  - Only API Gateway, with firewall, should be accessed from outside
  - Every call needs to be authenticated with a token
  - Public and private keys need to be continuously rotated
  - 2FA should be an option for customers and requirement for internal users
  - Service->Service calls need to be secured to prevent unintentional exposure

Burning money at scale when demand hits

- What do you really do when demand goes up 5-10x overnight?
- Answer: **burn money** by scaling up your infrastructure.
- Do all of these things:
  - Increase number of containers and VMs
  - Increase instance sizes and resources available to containers
  - Upgrade to bigger and faster Redis instances
  - Spend more $$$ on your SNS / SQS / Kafka / Kinesis / MQ messages
  - Increase your serverless lambda budget and start looking into replacing lambdas with workers
  - Realize that any piece of your infrastructure that depends on pay-per-request (AWS API Gateway, WAF, X-Ray, DynamoDB on-demand, etc.) now costs you a small fortune and you need to start planning on replacing it

When to scale? Monitoring and Alerts

- Watch out for premature scaling, i.e. “spending money on cloud resources beyond the need of supporting your customers”
- Resource (CPU, memory, storage) utilization alerts need to be set for all datastores, queues, and services
- Service runbooks need to be thought out in advance on what actions to take in cases of increased demand
- Runbooks need to be tested in staging environment
- Chaos Engineering needs to be a regular practice



Conclusion

- Cloud computing provides great ability to make your services up and running quickly without initial infrastructure investments
- Horizontal scaling in the cloud allows you to handle increased demand and grow your business
- Engineers need to build systems that can scale horizontally
- Continuous review and search for solution to reduce infrastructure cost is required; otherwise, there is a high risk to go bankrupt