Serverless Basics

→ Frank Siler me@franksiler.com

2021-09-16

#include <disclaimer.h>

So what is this “Serverless” thing, anyway?

My definition: Serverless is a set of principles and technologies to allow the construction of software which is

- highly scalable
- highly available
- cost-manageable
- decomposable
- geographically distributed
- capable of being monitored

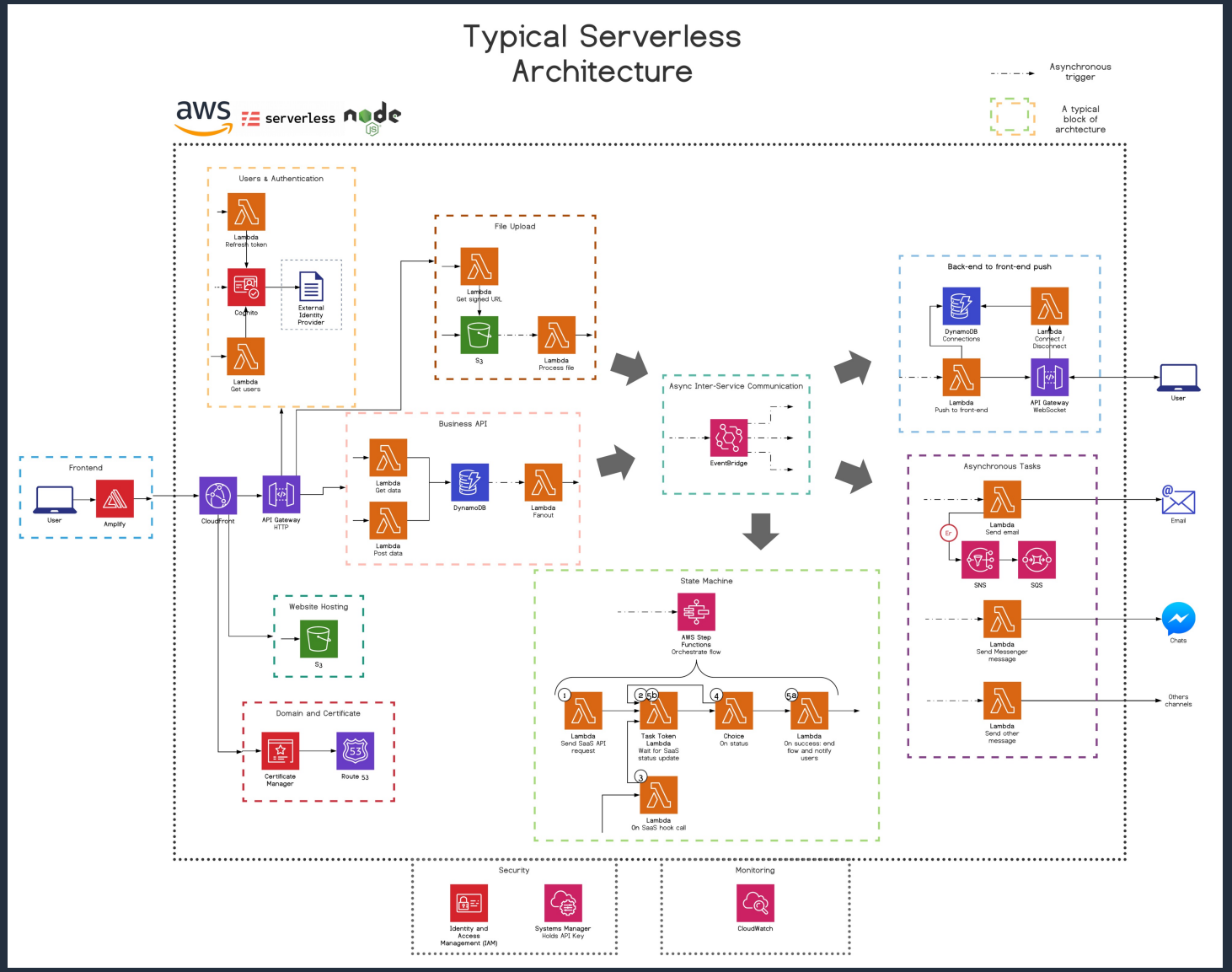
How do you go about that?

Thoughtworks laid out these ideas. Some of the important characteristics of serverless systems are:

- Low barrier to entry
- Hostless
- Stateless
- Elastic
- Distributed
- Event-Driven

https://www.thoughtworks.com/en-us/insights/blog/traits-serverless-architecture

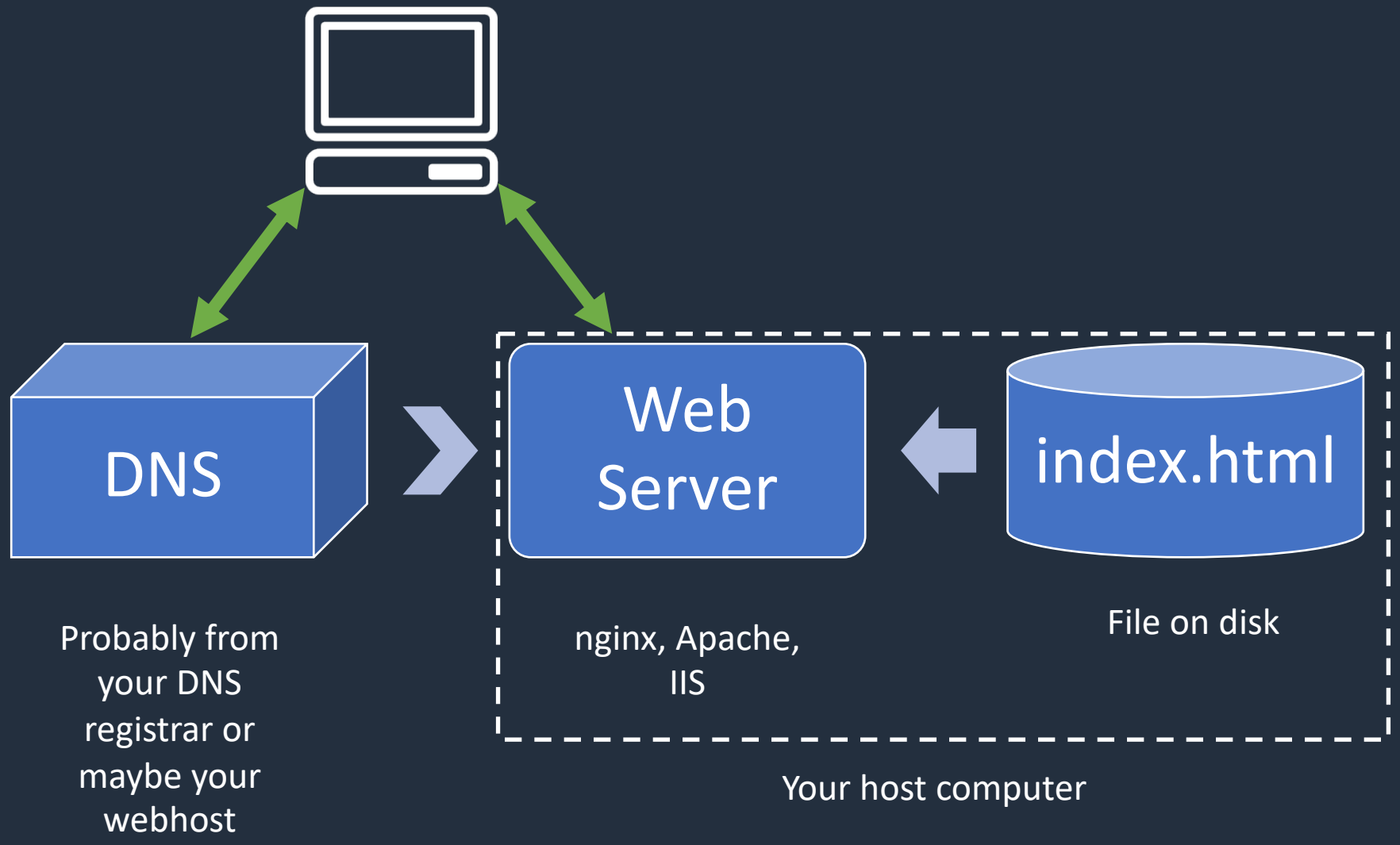
It doesn’t have to be complicated…..



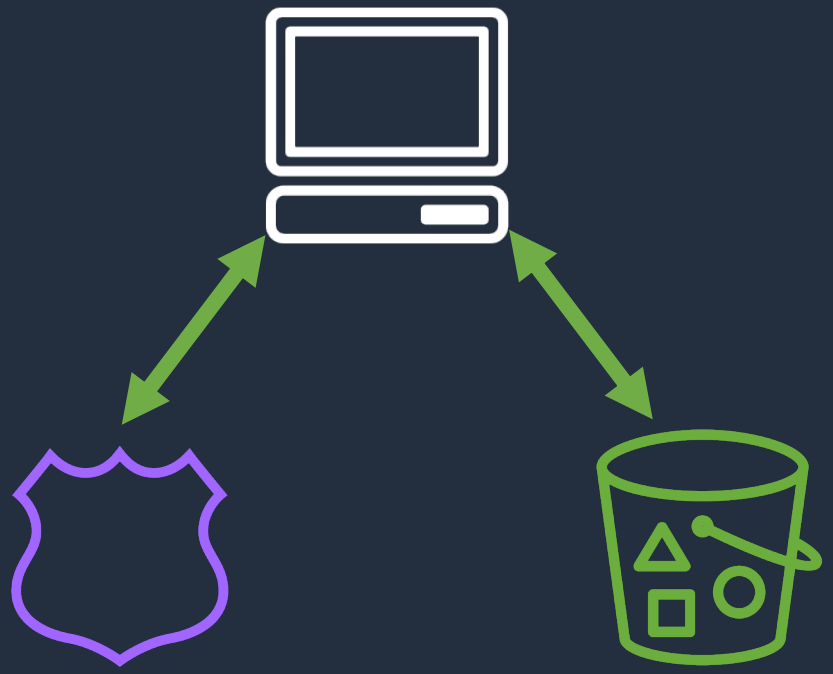
https://medium.com/serverless-transformation/what-a-typical-100-serverless-architecture-looks-like-in-aws-40f252cd0ecb

How would you do it in the old days?

A simple static website:



The Serverless Version



Cloud DNS provider, eg Route 53

Cloud Storage Provider eg S3, GCS, Azure Storage

So what's the big deal?

- You didn't change much!
- Exactly. Except that instead of running on one computer, we are now running on a vast distributed network.
- For many applications, this will cost you less and have higher uptime than the old model. BUT…
- An aside. Cloud is not ideal for everything. It has significant advantages for workloads that are:
  - Bursty or unpredictable
  - Experimental
  - Geographically distributed
- You must do your own math and understand your own application to make an intelligent decision. Don't be breathless about any given technology.

Quick Commentary on the Big Three

GCP: most developer friendly

- Upside: Super cool databases: BQ, Spanner, BigTable
- Downside: losing money, not platform company / deprecation policy

Azure: obvious choice if you're a Microsoft shop, particularly for MSSQL

- Upside: great integration for Windows stuff but also OSS friendly, privacy
- Downside: cost

AWS: suggest choosing only if they have a specialty service you need

- Upside: lots of specialty services (eg, Satellite Downlink)
- Downside: janky. Poor documentation. A friend of a friend was the sole responsible person for EC2 for 12 hours a day.

So what technologies do we use?

- Technologies to **connect**
- Technologies to **remember**
- Technologies to **process** and **understand**
- Technologies to **diagnose** and **manage**

Technologies to connect

- HTTP gateway
- CDN
- Subnet
- VPN
- Firewall
- Site-to-site connection
- Message queues
- Notification system

Technologies to remember

- Object storage / artifact repository
- Caching databases: memcache, redis
- Distributed databases: MongoDB, Cassandra, CouchDB
- Then the big boys- proprietary, globally distributed:
  - AWS: DynamoDB, RedShift
  - GCP: Firestore, BigTable, Cloud Spanner, BigQuery
  - Azure: CosmosDB
- And then possibly the most interesting: modern serverless databases with old-school APIs:
  - AWS: Aurora (MySQL and PostgreSQL), DocumentDB (MongoDB)
  - GCP: Atlas (MongoDB)
  - Azure: Azure SQL Database (MSSQL), Azure DB for PostgreSQL

Technologies to process and understand

- Functions and general compute
- Machine Learning
  - Sentiment analysis
  - Text and document processing
  - Image and video processing
  - Speech processing
  - Agents
  - Translation
- Search

Technologies to diagnose and manage

- Logging
- Tracing
- Monitoring
- Profiling
- User management
- Source control
- Deployment tools
- Cost management

Third Party products

- Auth0 – login
- Twilio – telephony

Limitations of serverless

- Your whole world generally belongs to someone else
  - Vendor lock-in
  - Latency, costs, tech support
  - The world can shift underneath you
- Can be annoying to debug
- Cold Starts
- Limited runtime support
- Per-unit costs (OpEx) tend to be higher
- Local dev can be more complex than more traditional setups
  - Fewer tools set up for it

https://www.oreilly.com/library/view/what-is-serverless/9781491984178/ch04.html

Some example applications

Example: S3 archiver

- So here's the thing: by default, you upload to the "Standard" storage class, which is $0.023/GB in us-east1. By contrast, "Deep Glacier" storage is nearly 25x cheaper
- But here's the thing: you pay for "operations", which include the copy that you need to change storage classes, and you also pay for 8KB of Standard storage and 32KB of Glacier for every object in Glacier.
- Therefore, it never makes sense to archive objects 8KB or smaller

Example: auto transcode on video upload

- Didn’t have a chance to get this up and running, but it’s the same kind of idea, except that we’re kicking off an outside task
- https://motorscript.com/automate-aws-elastic-transcoder-s3-files-using-lambda-functions/

Example web app: FatPitch

Questions?

Resources

- Possibly my favorite introductory talk: Sam from ACloudGuru https://youtu.be/Cd0qLRkTubk
- For the old-fashioned (but still valid) way:
  - Hosting deals aggregator: lowendbox.com
  - Hosting: ssdnodes.com
- https://martinfowler.com/articles/serverless.html
- https://www.oreilly.com/library/view/what-is-serverless/9781491984178/
- Interesting blogs
  - http://rachelbythebay.com/w/
  - https://boyter.org/posts/abusing-aws-to-make-a-search-engine/