A CLOUD GURU

SERVERLESS TALES FROM THE TRENCHES
After three days at @Serverlessconf I can see that new world is not only possible, it's on it way. Biggest takeaway, early adopting teams of #cloud and now #serverless are iterating and adapting exponentially faster than industry. We know how this story ends, who wins and loses.

Simon Wardley
@swardley

"The proverbial shit will hit the fan when a two-person company that produces a single function that everyone uses gets acquired for $1Bn. It's only a matter of years" - sf.serverlessconf.io/hackathon.html
Peter Sbarski, PhD
AWS Serverless Hero
@sbarski

Author
Serverless Architectures on AWS
https://book.acloud.guru

VP Engineering | VP Content
A Cloud Guru
https://acloud.guru

Organizer
Serverlessconf
https://serverlessconf.io
TEACHING THE WORLD TO CLOUD

Join the 600,000+ engineers that have taken courses with A Cloud Guru.
## Why Serverless?

<table>
<thead>
<tr>
<th></th>
<th>IaaS</th>
<th>PaaS</th>
<th>Serverless</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit of Scale</strong></td>
<td>Virtual Machine or Container (Docker)</td>
<td>Application</td>
<td>Function</td>
</tr>
<tr>
<td><strong>Fleet Operational Responsibility</strong></td>
<td>Application developer</td>
<td>Shared between developer and vendor</td>
<td>Vendor only</td>
</tr>
<tr>
<td><strong>Required Management &amp; Maintenance</strong></td>
<td>High – Operating System level</td>
<td>Medium – Application level</td>
<td>Low – function level</td>
</tr>
<tr>
<td><strong>Billing</strong></td>
<td>Per VM per minute or hour</td>
<td>Per VM per minute or hour</td>
<td>Per 100 milliseconds (continuous)</td>
</tr>
<tr>
<td><strong>Impact of Idle Time</strong></td>
<td>Economic loss when machines are idle or underused</td>
<td>Economic loss when machines are idle or underused</td>
<td>None – functions execute only when needed</td>
</tr>
<tr>
<td><strong>Integration with other vendor services</strong></td>
<td>Manual</td>
<td>Mixed</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

Source: Serverless Design Patterns (T. Wagner, Y. Kiriati, P. Sbarski)
July 2018 – Lambda

107,836,011 invocations of Lambda per month

45,796,766 million API invocations per month
July 2018 – AWS

Footprint:
- 289 Lambda Functions
- 19 Micro-services
- 3.68TB of data in S3
- 107m Lambda Invocations
- 45m API Requests
- 3.8+ TB of data via CloudFront per day
- 650K+ users
No Ops? Yes, Ops.
Automation is not optional
Serverless Stories

Frontend and API Failover

Making functions & services more resilient

Patterns and architectures
Frontend Failover
Frontend Failover

December 2017

Website Outage
The website has been stable now for some time so we are marking this issue as resolved. We will update this status later.
Dec 13, 10:34 - 14:15 AEDT

November 2017

No incidents reported for this month.

October 2017

No incidents reported for this month.
Frontend Failover

Health Check

Route 53

User

Primary Website

CloudFront

S3 Bucket

Backup Website

Check health every few seconds
Frontend Failover
Frontend Failover

Danger zone
Irreversible and destructive actions. Tread lightly.

Delete site
Once you delete a site, there is no going back.

Delete this site
API Failover
API Failover

API Gateway – US East 1

API Gateway – US West 1
API Failover
API Failover
API Failover
Making functions & services more resilient
Handling Errors

Peter uploads a file

Source S3 Bucket

Lambda

Elastic Transcoder

Firebase (Database)

Destination S3

Lambda

SNS

Lambda

Destination S3
Handling Errors

Peter uploads a file
Source S3 Bucket
Lambda
Elastic Transcoder
Dead Letter Queue (SNS)
Error!
Firebase (Database)
Lambda
Error!
SNS
Destination S3
Burning down the house
Burning down the house

You might still over provision or under provision DB connections

Multiple functions may need DB access with different usage profiles at different times.
Burning down the house

Dynamic Connection Management

Do something interesting → CloudWatch Alarm → Amazon CloudWatch

Amazon DynamoDB

Maintain atomic counter and max count

Helper

Open / close connections

DB operations

https://github.com/aws-samples/aws-lambda-manage-rds-connections
Tips

• 1 function = 1 task (avoid fat monoliths)
• No state (be idempotent)
• Design for failure
• More memory = more CPU and IO
• Set function concurrency to 0 as a kill switch
• Keep permissions and roles tight
• Beware of Serverless cancer (try to avoid recursion)
• Serverless monoliths can be OK initially!
• Incremental architecture is not dirty
• Use Step Functions or Durable Functions for workflows
Patterns and Architectures
Patterns

**Primitive**
- Periodical (Cron Jobs)

**API**
- Proxy
- Facade

**Orchestration**
- One way chain
- Two way chain
- Fan in
- Fan out

**Workflows**
- Long Running tasks
- Pipes and Filters
- Inline Stream Transform

**Traditional**
- Command
- Singleton

**Compound**
- Backends
- CQRS
- Data processing

Source: Serverless Design Patterns (T. Wagner, Y. Kiriaty, P. Sbarski)
**Patterns**

**Name**
API Proxy (also known as wrapper)

**Description**
Acts as a mediator between two systems that cannot communicate directly. Transforms request and response payloads to facilitate exchange of information.

**Motivation**
Useful when incompatible systems need to talk. Reduces coupling by removing the need to build direct dependencies between incompatible systems.
Transformer function transforms JSON to XML and back again.

System A (JSON)  →  API Gateway  →  Transformation Function  →  System B (XML)
Patterns

Name
Simple fan-out

Description
Allows multiple endpoints to receive a copy of an input event. Turns any single-receiver delivery system into a multiple-receiver system.

Motivation
Event-based systems are often designed to have a single receiver for events, and API calls are by definition single receiver. The simple fan-out pattern asynchronously delivers its triggering event to one or more workers.
Simple fan-out

How would you design transactional fan-out?
Patterns

Name
Inline stream transform

Description
Transmits data between systems. Can be chained, can multiple and demultiplex at the source or destination. A transformation function can transform a record with the result progressing through the stream.

Motivation
A way to decouple systems that share data. Can offer temporal decoupling by allowing producers and consumers to operate at different rates. A transform function is used to clean, modify, group, analyze data before it gets to the consumer.
A stream can be sharded if order is not important.
How can you get started?

- serverlessconf.io & video.serverlessconf.io
- Follow @serverlessconf for serverlessconf info
- A Cloud Guru acloud.guru/serverless
- Follow @acloudguru and @sbarski
Thanks :-)

https://acloud.guru
https://serverlessconf.io
@sbarski