Building Bespoke Threat Intelligence Enrichment Platforms

Prepared by: Sean Tierney

Date:
Serverless Architecture

Discussion points:

- Definition and usage
- Example
- Benefits and drawbacks
- Questions
#!/bin/bash
for feed in (cat /usr/feedz/srcs)
  do
curl -K $feed >> output.tmp
fi
cat output.tmp | sort –u > threats.txt
Definition and Usage

How did it start?
Serverless architecture emerged as a viable cloud application architecture in 2014 with the launch of Amazon’s Lambda service.

What makes Serverless architecture unique?
As the name would imply, all web servers, databases, and application servers are abstracted away from the developer.
More than just running code: Database solutions, API Gateways, Messaging, etc.
Trends
Harvester simplified

Submitted Data → Validation

Data egress

Enrichment → Data storage
API Gateway

Used for:

- Managing API access

User Request

API layer

Auth hooks

Validation

Amazon API Gateway
Lambda

Used for:

- Data validation and collection
- Database access
- Data egress

Validation ➔ Groomer ➔ Egress ➔ Data collection
Dynamo DB

Used for:

- Data storage
- Global state control
- Event triggers

Diagram:

- Data Collection
  - AWS DynamoDB

- Groomer
  - AWS DynamoDB

- Modified Event

- Data Table

- Enrichment Table
S3

Used for:

- Archiving
Cloud Formation

Used for:

• Setup / Teardown
• Deployments
Benefits
Faster time to market

- No patching or upgrading of servers required
- Fewer configurations to define - less training required
- Allows for better separation of business logic from infrastructure
- Developers get code working faster
Scalability
Cost

- Relatively inexpensive code executions
- Pay for only what you use
- Easier to learn and faster to develop which means fewer developer hours are required
Drawbacks
Vendor lock-in

- Operate via proprietary APIs
- Small number of providers
- Existing projects may not be compatible with other providers - no agreed upon features list.
- Application is tied to the lifespan of the platform (and their price increases)
Performance Issues

- Code that is not run often incurs a startup penalty
- No globally accessible server state means that more database or storage calls may be required.
- Services are not as configurable as a server. Code is generally run on a generic configuration.
Not mature yet

- Developers with experience are harder to find (but easier to train)
- No universally accepted best practices
- Limited number of programming languages
- Security?
Tooling

- Fewer options for application management
- Deployment tools cannot transfer to other platforms and may not provide all the features needed.
- Logging can be a mess
Cost

- Expense is not static - scales with the application
- Pricing model is complicated - total price is difficult to predict
- Significant refactoring cost to switch platforms
Title here (Arial Regular 28pts)

• Level 1 text (Arial Regular 20pts)
  – Level 2 text (Arial Regular 18pts)
    ◦ Level 3 text (Arial Regular 16pts)
      • Level 4 text (Arial Regular 12pts)
Bar and Pie Chart
Bar Chart

Chart Title

Category 1
Category 2
Category 3
Category 4

Series 1
Series 2
Series 3
Transition slide