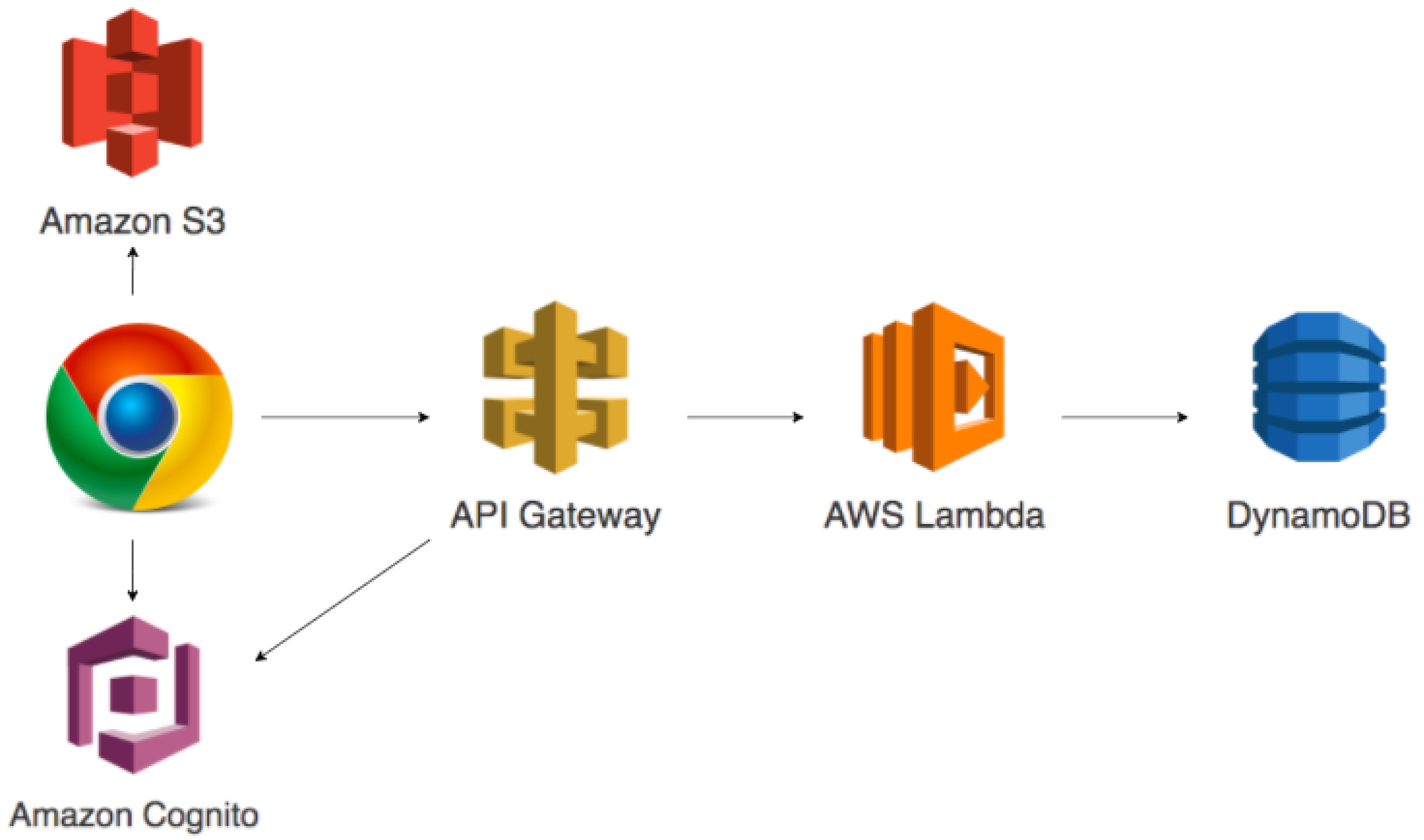
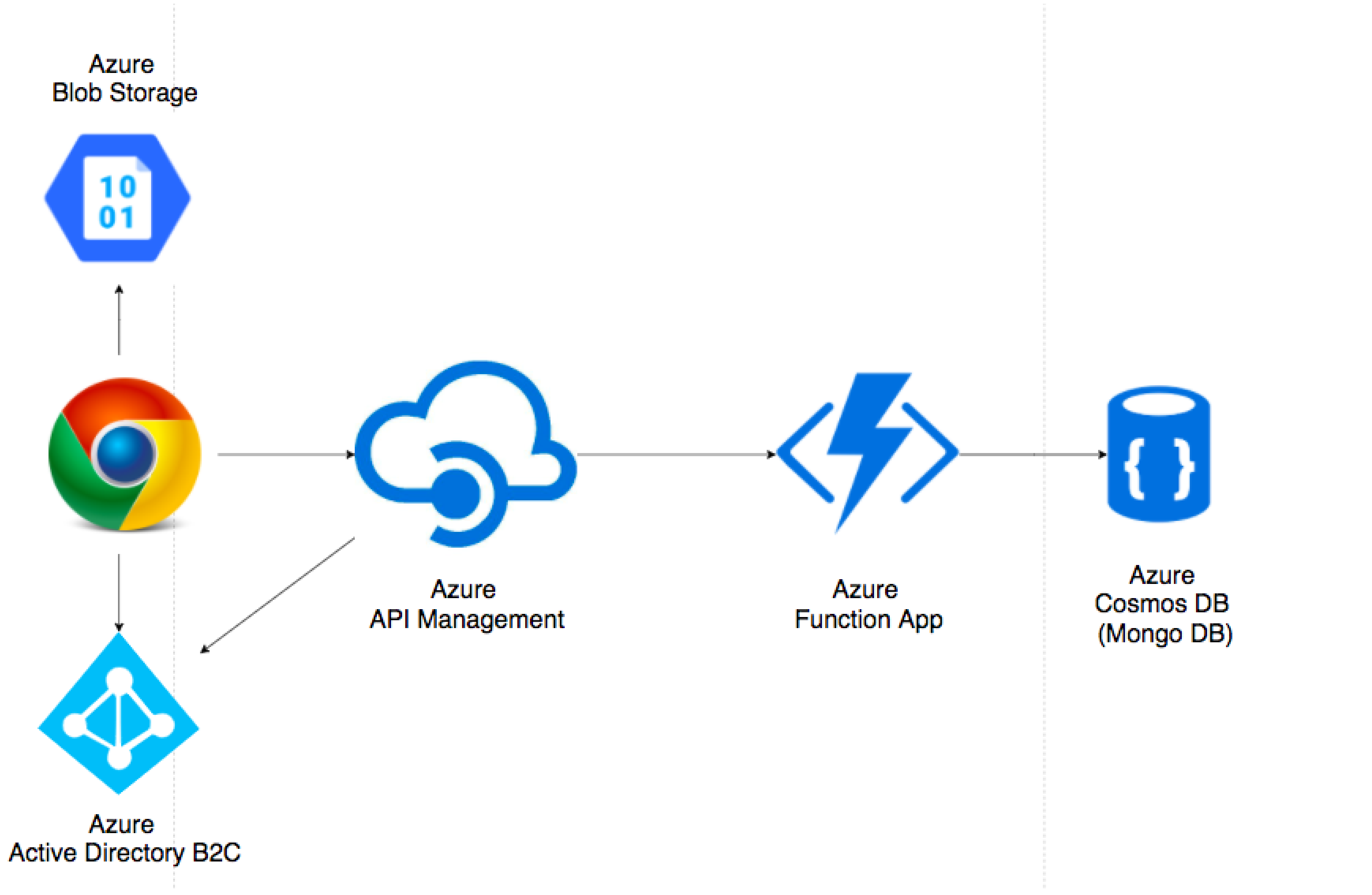
Go Serverless with AWS & Azure

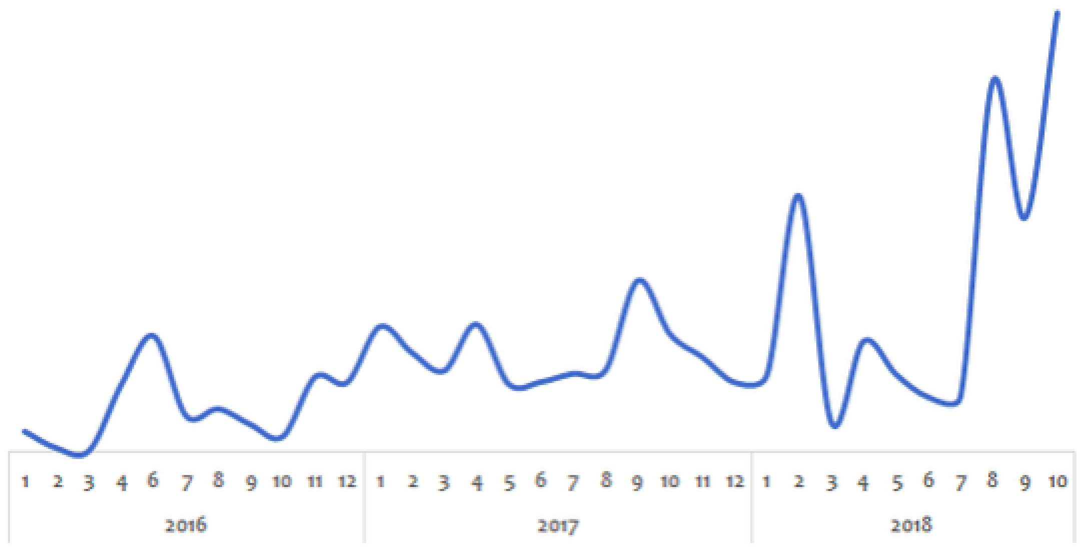
AWS Serverless Architecture



Azure Serverless Architecture

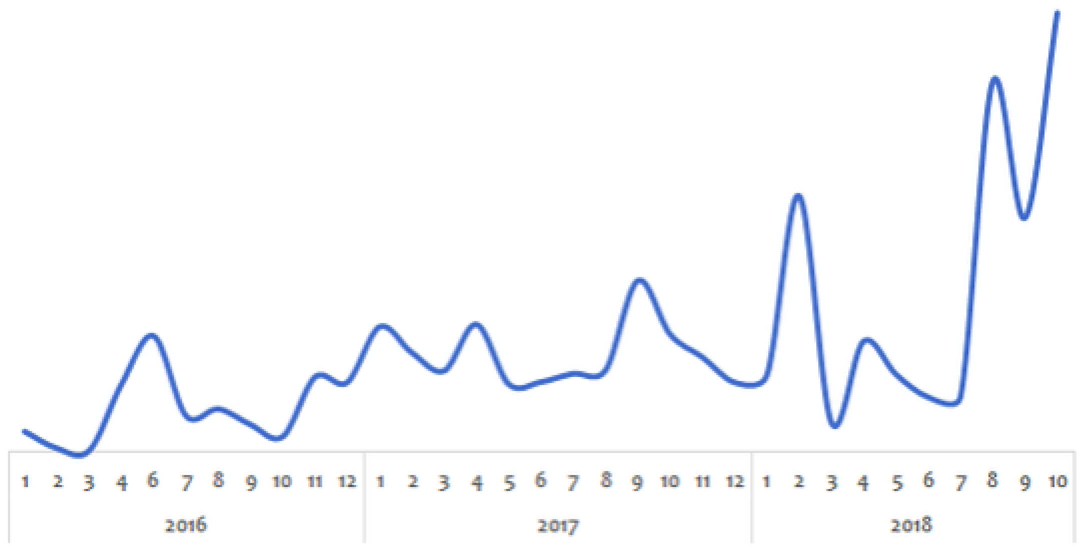


Before the Cloud



- Consider a Online Shopping Application:
  - Peak usage during holidays and weekends. Less load during rest of the time.
- A startup suddenly becomes popular:
  - How does it handle the **sudden increase** in load?
- Enterprises **procured** (bought) infrastructure **for peak load**
  - Startups procured infrastructure assuming they would be successful

Before the Cloud - Challenges



- Low infrastructure utilization
- Needs ahead of time planning (**Can you guess the future?**)
- High costs of procuring infrastructure
- Dedicated infrastructure maintenance team (**Can a startup afford it?**)

Silver Lining in the Cloud

- How about **provisioning (renting) resources** when you want them and releasing them back when you do not need them?
  - **On-demand resource provisioning**
- Advantages
  - Lower costs (Pay per use)
  - No upfront planning needed
  - Avoid **"undifferentiated heavy lifting"**
- Challenge
  - Building cloud enabled applications

Amazon Web Services (AWS)

- Leading cloud service provider
- Provides most (200+) services
- Reliable, secure and cost-effective

Serverless

AWS Lambda API Gateway DynamoDB

- What are the things we think about when we develop an application?
  - Where do we deploy the application?
  - What kind of server? What OS?
  - How do we take care of scaling the application?
  - How do we ensure that it is always available?
- **What if we do not need to worry about servers and focus on building our application?**
- Enter **Serverless**

Serverless

- Remember: **Serverless does NOT mean "No Servers"**
- **Serverless for me:**
  - You don't worry about infrastructure
  - Flexible scaling
  - Automated high availability
  - Pay for use:
    - You don't have to provision servers or capacity!
- **You focus on code** and the cloud managed service takes care of all that is needed to scale your code to serve millions of requests!

AWS Lambda

- Write and Scale Your Business Logic
  - Write your business logic in Node.js (JavaScript), Java, Python, Go, C# and more..
  - Don't worry about servers or scaling or availability (only worry about your code)
- Pay for Use
  - Number of requests
  - Duration of requests
  - Memory

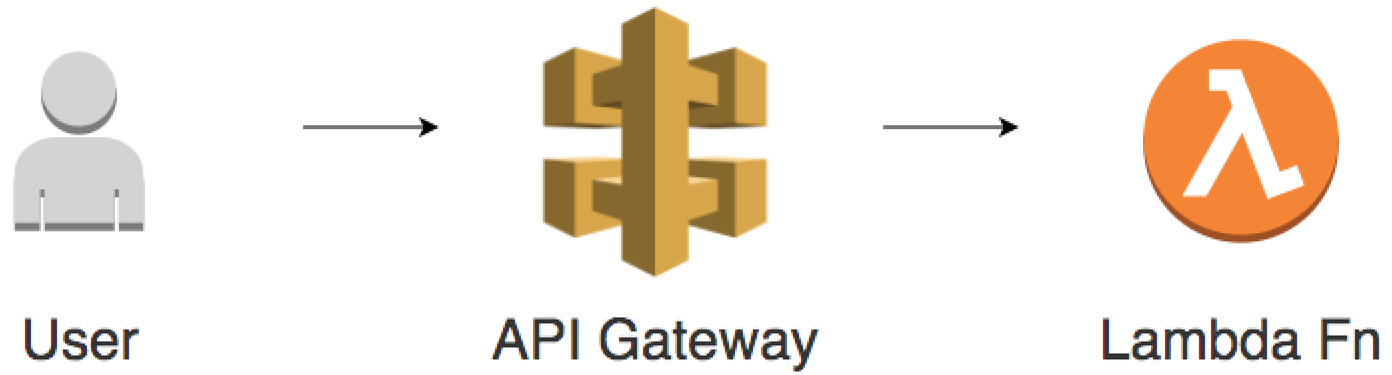
Lambda Function Concurrency

In28 Minutes

- **Function concurrency** - no of Lambda function instances serving requests (at a given time)
- You invoke a lambda function for the first time:
  - AWS Lambda creates an instance of the function
  - AWS Lambda runs the function's handler method and returns the response
  - (However) The function instance stays active and waits to process new events
- You invoke the lambda fn again (while first event is in progress):
  - AWS Lambda creates another instance of the function
- When no of requests decreases, Lambda stops unused instances
- (However) There are limits to concurrency of lambda function (DEMO)
- Provisioned concurrency can help to avoid fluctuations in latency (DEMO)

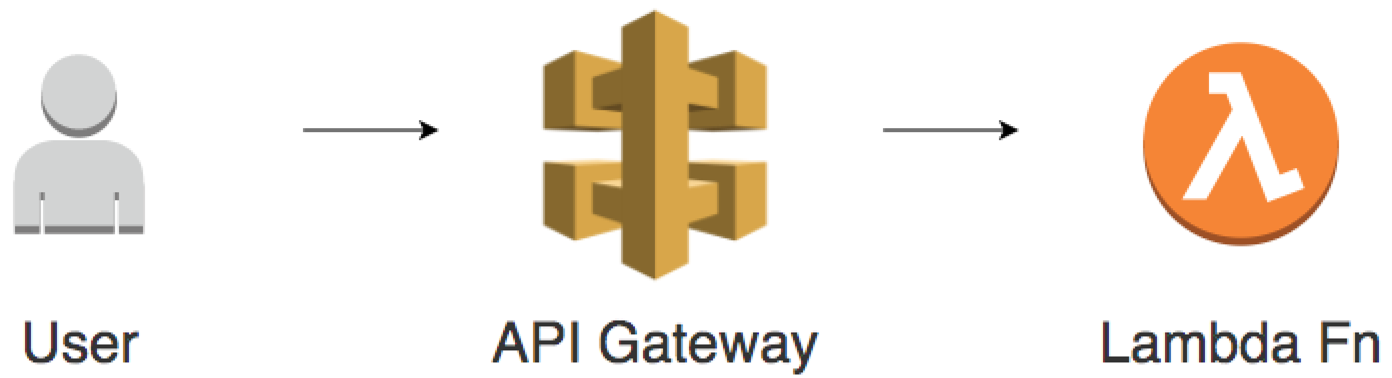
AWS Lambda

REST API Challenges



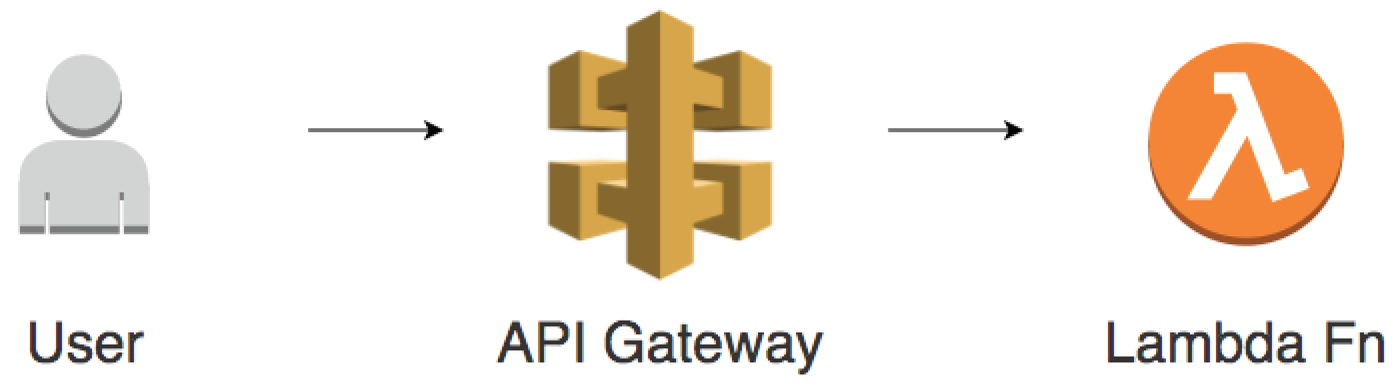
- Most applications today are built around REST API:
  - Resources (/todos, /todos/{id}, etc.)
  - Actions - HTTP Methods - GET, PUT, POST, DELETE etc.
- Management of REST API is not easy:
  - You've to take care of authentication and authorization
  - You've to be able to set limits (rate limiting, quotas) for your API consumers
  - You've to take care of implementing multiple versions of your API
  - You would want to implement monitoring, caching and a lot of other features..

Amazon API Gateway



- How about a **fully managed service** with auto scaling that can act as a **"front door"** to your APIs?
- Welcome **"Amazon API Gateway"**

Amazon API Gateway

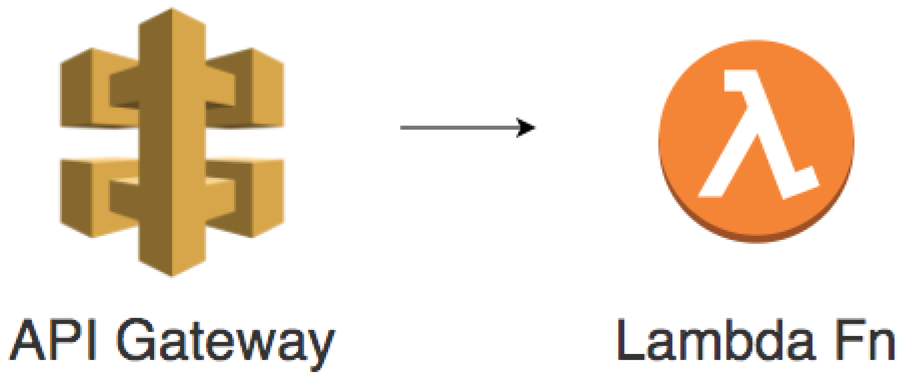


- **"publish, maintain, monitor, and secure APIs at any scale"**
- Integrates with AWS Lambda or any web application
- Supports HTTP(S) and WebSockets (two way communication - chat apps and streaming dashboards)
- Serverless. **Pay for use** (API calls and connection duration)

API Gateway - API Types

- **REST API**
  - feature-rich RESTful API
- **HTTP API**
  - Also used to build RESTful API
  - Newer approach
- **WebSocket API**
  - Persistent connections with clients
  - Allows full-duplex communication
- Names are little confusing

API Gateway RESTful API approaches



REST API

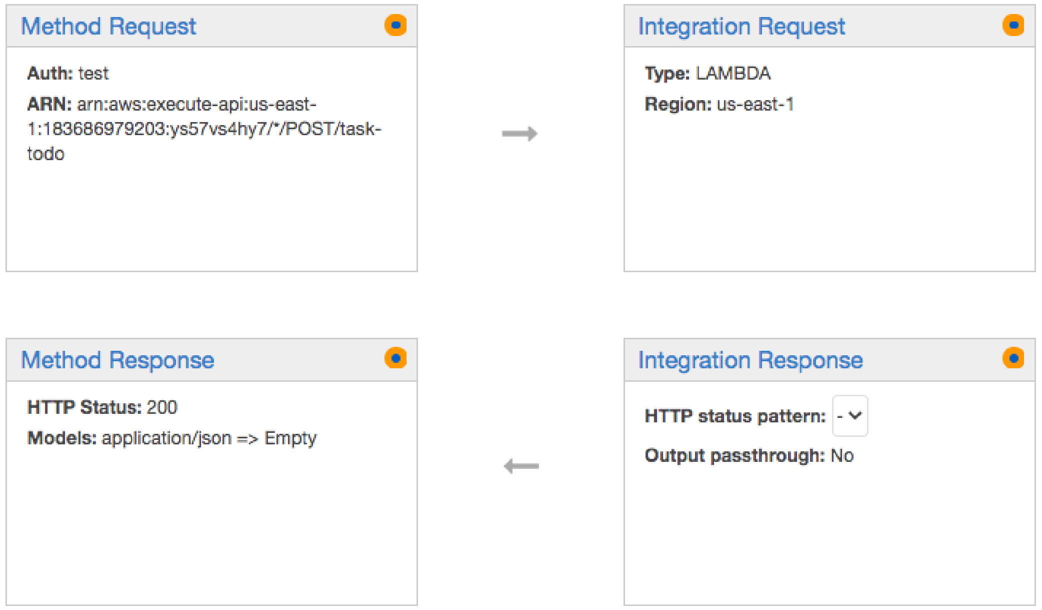
- Fully Featured (API caching, Request/Response Validations, Test invocations)
- Custom Request/Response Transformations
- Better Integration with AWS Services (AWS X-Ray, AWS WAF etc)

HTTP API

- Newer, Simpler, Cheaper and Low Latency
- Automatic Deployments

REST API Gateway - Custom Integration (Default)

- Integrations define request/response transformation to/from lambda
- The default is Custom Integration
- Request can be transformed by configuring Mapping Template in Integration Request
- Response can be transformed by configuring Mapping Template in Integration Response



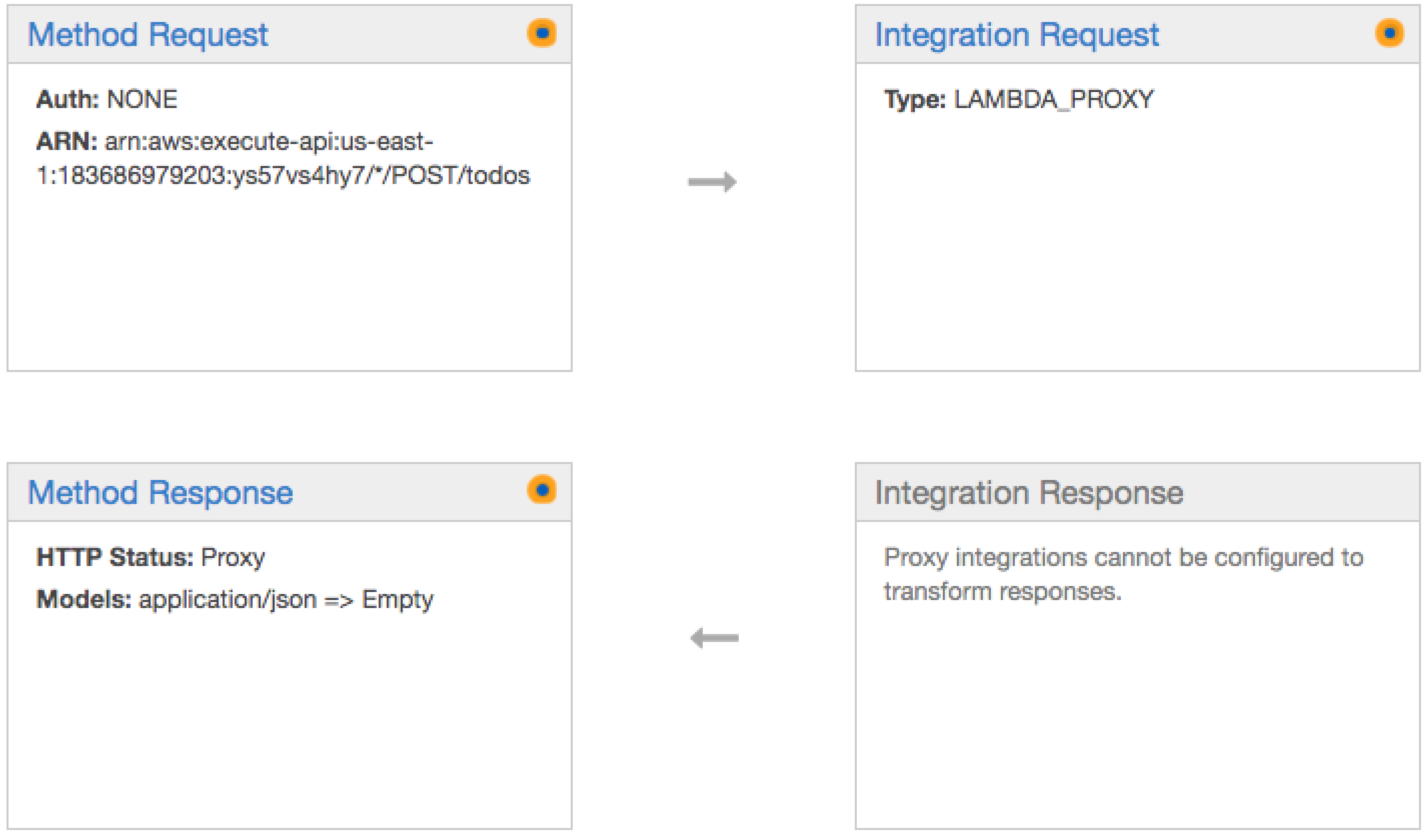
Auth: test
ARN: arn:aws:execute-api:us-east-1:183686979203:ys57vs4hy7/*/POST/task-todo

Type: LAMBDA
Region: us-east-1

HTTP Status: 200
Models: application/json => Empty

HTTP status pattern: -
Output passthrough: No

REST API Gateway - Proxy Integration



Auth: NONE
ARN: arn:aws:execute-api:us-east-1:183686979203:ys57vs4hy7/*/POST/todos

Type: LAMBDA_PROXY

HTTP Status: Proxy
Models: application/json => Empty

Proxy integrations cannot be configured to transform responses.

- How about defining a standard transformation?

REST API Gateway - Proxy Integration - Request

Request to API Gateway

```
//Headers: header1:header-value
//queryString: ?queryparam=queryparamvalue
{
    "message": "Welcome"
}
```

Standard event sent to Lambda Function

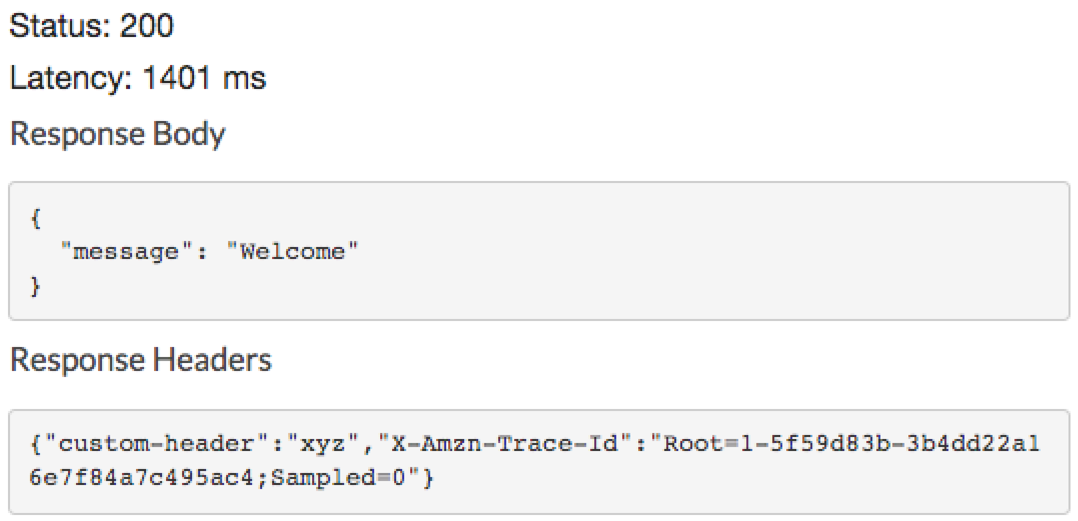
```
{
    resource: '/todos',
    path: '/todos',
    httpMethod: 'POST',
    headers: {"header1":"header-value"},
    multiValueHeaders: {"header1":["header-value"]},
    queryStringParameters: {"queryparam":"queryparamvalue"},
    multiValueQueryStringParameters: {"queryparam":["queryparamvalue"]},
    pathParameters: null,
    stageVariables: null,
    requestContext: {},
    body: '{\n  "message" : "Welcome"\n}',
    isBase64Encoded: false
}
```

REST API Gateway Proxy Integration - Response

Response from Lambda Function

```
{
    statusCode: 200,                        // a valid HTTP status code
    headers: {
        custom-header: "xyz"                // any API-specific custom header
    },
    body: "{\"message\": \"Welcome\"}"    // a JSON string.
}
```

Response from API Gateway



Status: 200

Latency: 1401 ms

Response Body

```
{
  "message": "Welcome"
}
```

Response Headers

```
{"custom-header":"xyz","X-Amzn-Trace-Id":"Root=1-5f59d83b-3b4dd22a1
6e7f84a7c495ac4;Sampled=0"}
```

HTTP API - API Gateway

- REST API - API Gateway has a lot of features very few AWS customers made use of
- REST API - API Gateway is a little complex to setup (transformations etc)
- How about creating a simpler API Gateway?
- Enter "HTTP API"
  - The naming is confusing
  - Newer, Cheaper and Low Latency
  - Simpler
    - Lesser features
    - Easier to setup
    - Example: Makes OAuth Authentication simple

API Gateway

HTTP API - API Gateway - Payload

- Two Versions - 1.0 and 2.0
- (Recommendation) Use 1.0 for migration from REST API and 2.0 for newer APIs
- Request Structure
  - Almost same as REST API - Proxy Integration
  - 2.0 offers support for cookies and has minor changes
- Response Structure
  - Same as REST API - Proxy Integration (with `statusCode`,`body`, `headers`)
  - In addition, 2.0 supports a simple structure:
    - Just return a valid response JSON `return {"message": "Welcome"}`

Amazon Cognito

- Want to quickly add a sign-up page and authentication for your mobile and web apps?
- Want to integrate with web identity providers (example: Google, Facebook, Amazon) and provide a social sign-in?
- Do you want security features such as multi-factor authentication (MFA), phone and email verification?
- Want to create your own user database without worrying about scaling or operations?
- Let's go : Amazon Cognito

Amazon Cognito

Amazon Cognito - User Pools

- Do you want to create your own secure and scalable user directory?
- Do you want to create sign-up pages?
- Do you want a built-in, customizable web UI to sign in users (with option to social sign-in)?
- Create a user pool

Amazon Cognito

Amazon Cognito - Identity pools

- Identity pools provide AWS credentials to grant your users access to other AWS services
- Connect identity pools with authentication (identity) providers
  - Your own user pool OR
  - Amazon, Apple, Facebook, Google+, Twitter OR
  - OpenID Connect provider OR
  - SAML identity providers (SAML 2.0)
- Configure multiple authentication (identity) providers for each identity pool
- Federated Identity
  - An external authentication (identity) provider
  - ex: Amazon, Apple, Facebook, OpenID or SAML identity providers

Amazon DynamoDB

- Fast, scalable, distributed for any scale
- Flexible NoSQL Key-value & document database (schemaless)
- Single-digit millisecond responses for million of TPS
- Do not worry about scaling, availability or durability
  - Automatically partitions data as it grows
  - Maintains 3 replicas within the same region
- No need to provision a database
  - Create a table and configure read and write capacity (RCU and WCU)
  - Automatically scales to meet your RCU and WCU
- Provides an expensive serverless mode
- Use cases: User profiles, shopping carts, high volume read write applications

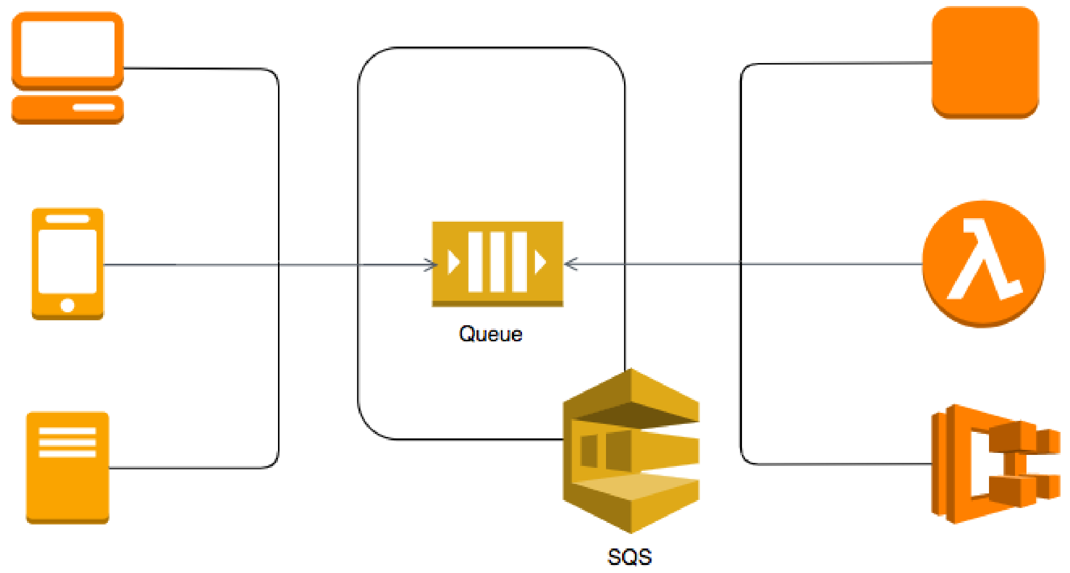
DynamoDB Tables

- Hierarchy : Table > item(s) > attribute (key value pair)
- Mandatory primary key
- Other than the primary key, tables are schemaless
  - No need to define the other attributes or types
  - Each item in a table can have distinct attributes
- Max 400 KB per item in table
  - Use S3 for large objects and DynamoDB for smaller objects

```
{
    "id": 1,
    "name": "Jane Doe",
    "username": "abcdefgh",
    "email": "someone@gmail.com",
    "address": {
      "street": "Some Street",
      "suite": "Apt. 556",
      "city": "Hyderabad",
      "zipcode": "500018",
      "geo": {
        "lat": "-3.31",
        "lng": "8.14"
      }
    },
    "phone": "9-999-999-9999",
    "website": "in28minutes.com",
    "company": {
      "name": "in28minutes"
    }
}
```

Simple Queuing Service

- Reliable, scalable, fully-managed message queuing service
- High availability
- Unlimited scaling
  - Auto scale to process billions of messages per day
- Low cost (Pay for use)



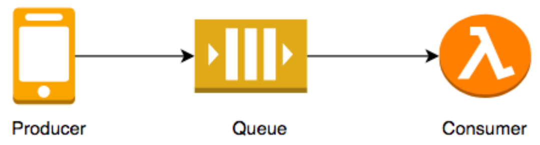
Standard and FIFO Queues

- Standard Queue
  - Unlimited throughput
  - BUT NO guarantee of ordering (Best-Effort Ordering)
  - and NO guarantee of exactly-once processing
    - Guarantees at-least-once delivery (some messages can be processed twice)
- FIFO (first-in-first-out) Queue
  - First-In-First-out Delivery
  - Exactly-Once Processing
  - BUT throughput is lower
    - Upto 300 messages per second (300 send, receive, or delete operations per second)
    - If you batch 10 messages per operation (maximum), up to 3,000 messages per second
- Choose
  - Standard SQS queue if throughput is important
  - FIFO Queue if order of events is important

Amazon SQS

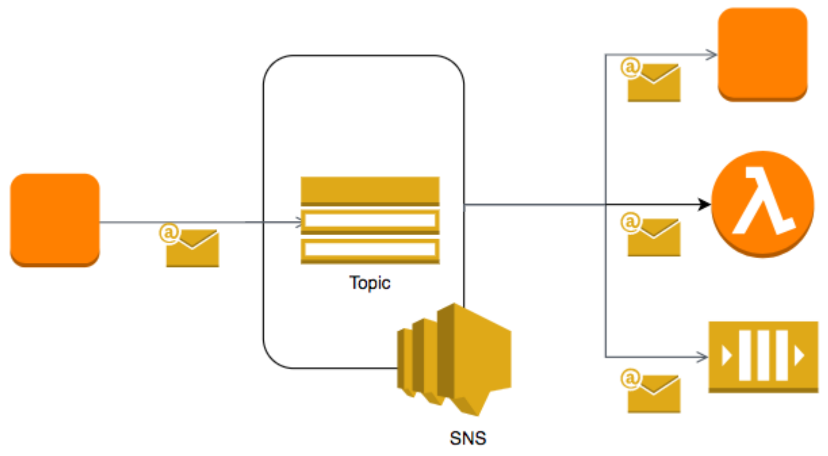
Sending and receiving a SQS Message - Best case scenario

- Producer places message on queue
  - Receives globally unique message ID ABCDEFGHIJ (used to track the message)
- Consumer polls for messages
  - Receives the message ABCDEFGHIJ along with a receipt handle XYZ
- Message remains in the queue while the consumer processes the message
  - Other consumers will not receive ABCDEFGHIJ even if they poll for messages
- Consumer processes the message successfully
  - Calls delete message (using receipt handle XYZ)
  - Message is removed from the queue



Amazon Simple Notification Service(SNS)

- Publish-Subscribe (pub-sub) paradigm
- Broadcast asynchronous event notifications
- Simple process
  - Create an SNS Topic
  - Subscribers can register for a Topic
  - When an SNS Topic receives an event notification (from publisher), it is broadcast to all Subscribers
- Use Cases : Monitoring Apps, workflow systems, mobile apps



Amazon Simple Notification Service(SNS)

- Provides mobile and enterprise messaging web services
  - Push notifications to Apple, Android, FireOS, Windows devices
  - Send SMS to mobile users
  - Send Emails
- REMEMBER : SNS does not need SQS or a Queue
- You can allow access to other AWS accounts using AWS SNS generated policy

Serverless Application Model

- 1000s of Lambda functions to manage, versioning, deployment etc
- Serverless projects can become a maintenance headache
- How to ensure that your serverless projects are adhering to best practices?
  - Tracing (X-Ray) etc
- Welcome SAM - Serverless Application Model
  - Open source framework for building serverless applications
  - Define a YAML with all the serverless resources you want:
    - Functions, APIs, Databases etc
  - BEHIND THE SCENES : Your configuration is used to create a AWS CloudFormation syntax to deploy your application

Serverless Application Model - References

Reference	Link
Developer Guide	*https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html*
AWS SAM Reference	*https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-reference.html*
AWS SAM Resource and Property Reference	*https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/sam-specification-resources-and-properties.html*

Serverless Framework

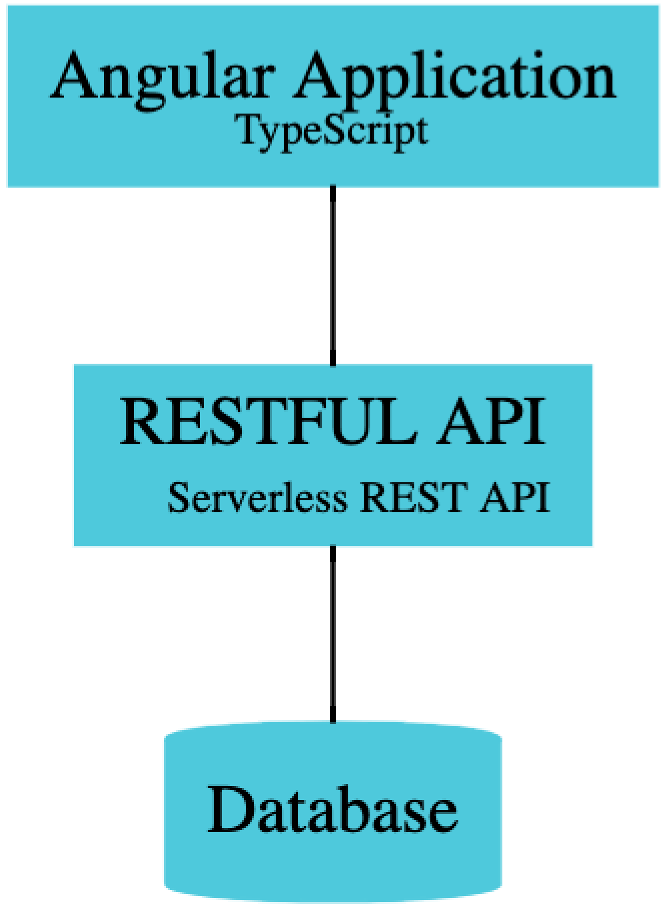
- 1000s of Lambda functions to manage, versioning, deployment etc
- Serverless projects can become a maintenance headache
- Welcome Serverless Framework
  - Zero-friction serverless development
  - Easy YAML + CLI development
  - Easy deployment to AWS, Azure, Google Cloud, etc
  - You can use a hosted dashboard (instead of CLI)

Serverless Framework References

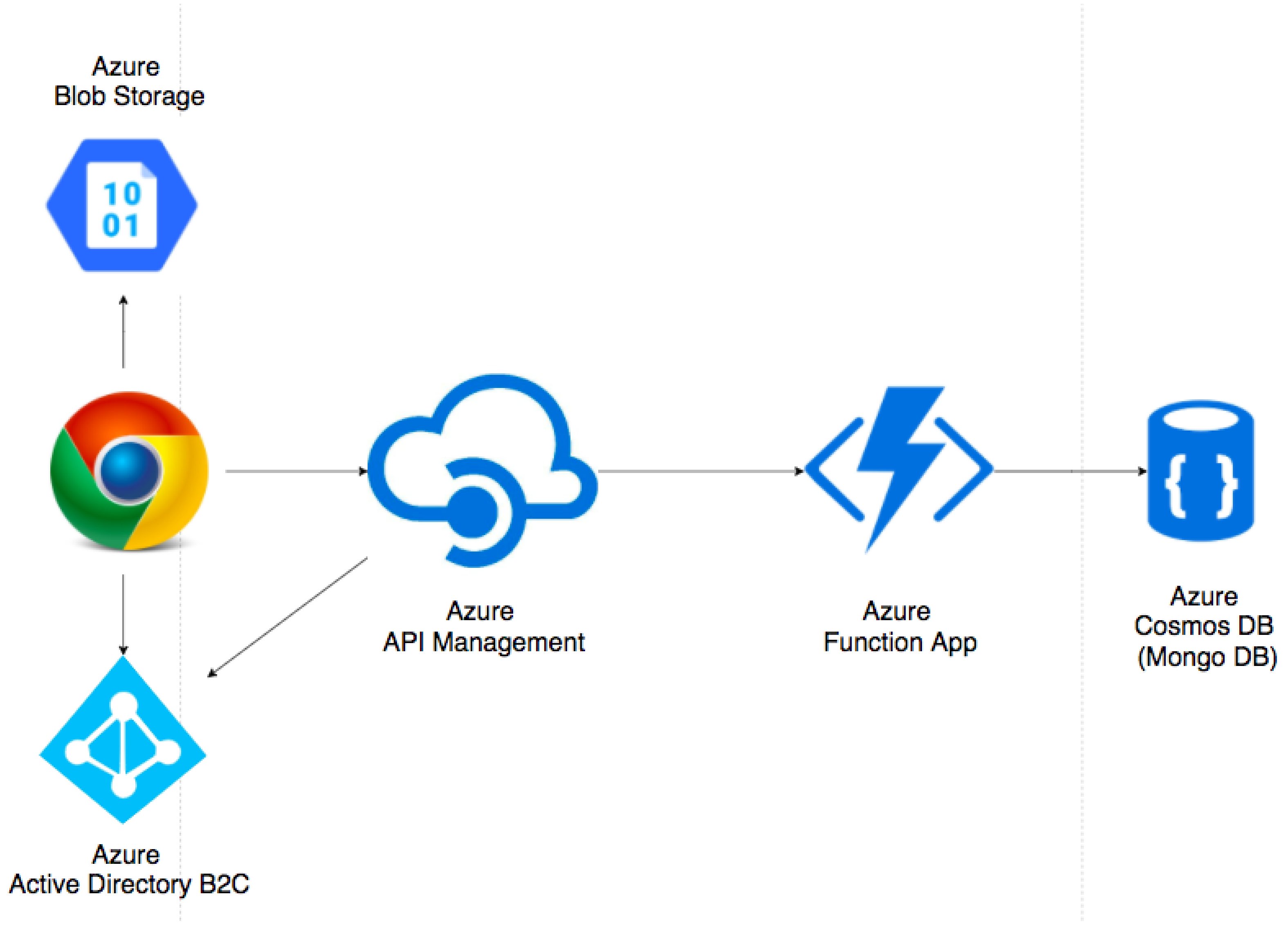
Reference	Link
Documentation	*https://www.serverless.com/framework/docs/*
Workflow Tips	*https://www.serverless.com/framework/docs/providers/aws/guide/workflow/*
Exploring Invoke Local	*https://www.serverless.com/framework/docs/providers/aws/cli-reference/invoke-local/*
serverless.yml template for AWS	*https://www.serverless.com/framework/docs/providers/aws/guide/serverless.yml/*

Full Stack Architecuture

- Architecuture
  - Frontend - Angular
  - Backend - REST API - Serverless or otherwise
- Setup
  - Node Js (npm)
  - Visual Studio Code
  - Angular CLI



Azure Serverless Architecture



You are all set!

Let's clap for you!

- You have a lot of patience! Congratulations
- You have put your best foot forward to get started with Serverless
- Keep Learning and
- Good Luck!

Do Not Forget!

- Recommend the course to your friends!
  - Do not forget to review!
- Your Success = My Success
  - Share your success story with me on LinkedIn (Ranga Karanam)
  - Share your success story and lessons learnt in Q&A with other learners!