

API Gateways and Microservices: 2 peas in a pod

Santosh Hari

Santosh Hari

Azure Consultant @ New Signature

Azure MVP

President, Orlando .NET UG

Organizer, Orlando Codecamp

 santoshhari.wordpress.com

 santosh.hari@newsignature.com

 [@_s_hari](https://twitter.com/_s_hari)

 [/in/santoshhari](https://in.linkedin.com/in/santoshhari)



NEWSIGNATURE

Agenda



Story of a monolith

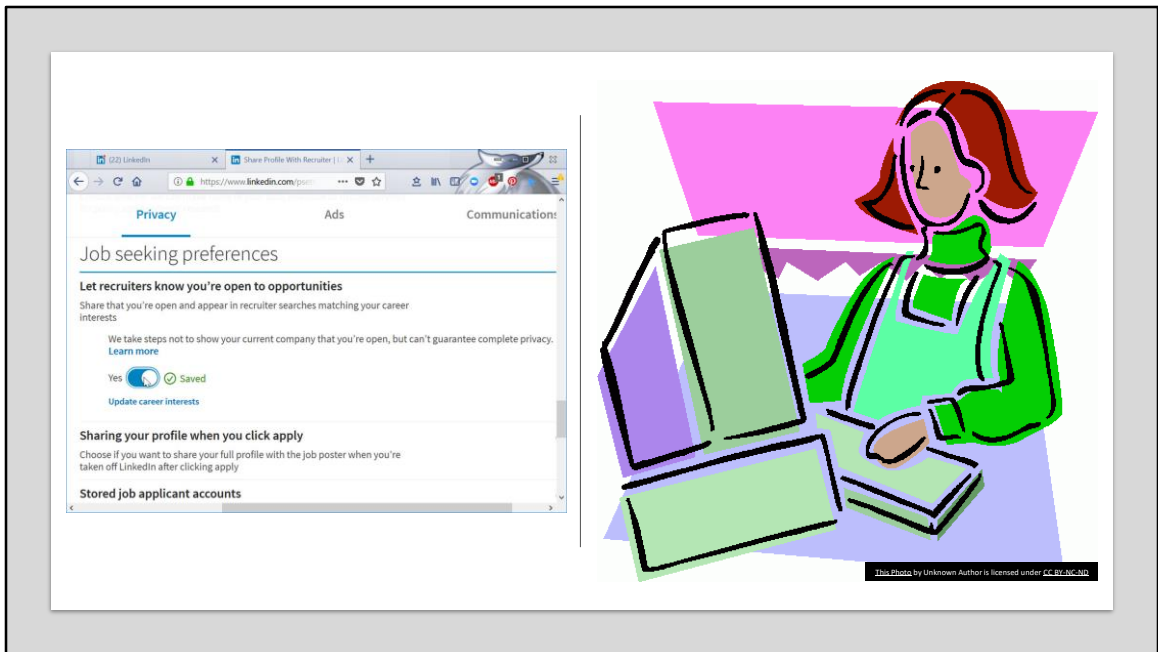


Monoliths to Microservices: An Unexpected Journey



API Gateways

Part 1: Story of a monolith



Another day, at the office,
You, the full stack dev working on "the next big thing."
your cellphone rings. It's your friendly recruiter - the one who calls you twice a day
about new jobs.
It's in some place in Wyoming, working on .NET 3.5 for \$15K/year
Not interested
Calls again tomorrow



Hot new startup w/ ping pong table, craft beer and stock options
This is an eCommerce startup: Like Uber but for pet products
Full online experience
Signup, Browse products, add to cart, checkout
So you do what you do best, build a website

- Website
 - Signup
 - Products
 - Checkout
 - Reviews
- Database

There are week long discussions on which Javascript framework to use – react, angular, vue
Then there are major discussions on what the backend code should be in – C#, Java
Then we start fighting about the database – sql server, mysql, oracle
Then there is a fight club organized for cloud provider – azure, aws, gcp
Someone mentions Oracle cloud and the entire team literally gangs up on them
Work starts in earnest, sprint after sprint,
Management and sales tag the product as behind schedule
But you know that you're killing it, delivering feature after feature,
The website looks great. You launch, there are issues but you fix and keep moving forward

Kiss of death for e-commerce applications



Then you get to thanksgiving.,

You discover that last minute your sales team decided to run a major discount for black Friday

As a result early morning on Black Friday your users start to see this – the kiss of death for ecommerce websites – the loading bar

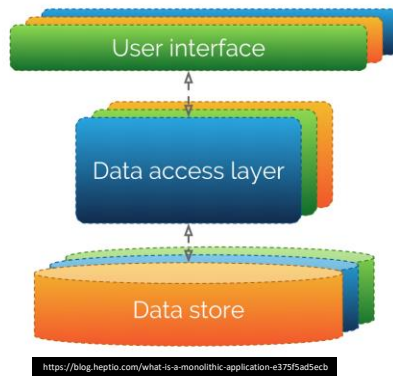
You scramble, since you're hosted on the cloud, you try to increase the capacity of your web and database

That works for a while until more and more users join and then it's back to the loading screen

And then users get frustrated and log off and suddenly the website is working again

And do back and forth it goes

Monolith



If you follow the guidance from past 15 years, you'll most likely build the system shown in Figure

Congratulations! You just created a monolithic application.

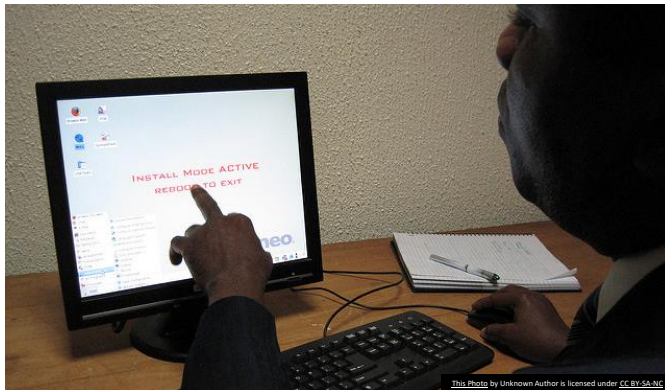
So you sit back and take stock of what you built.

You constructed a large core application containing all of your domain logic.

It includes modules such as Identity, Catalog, Ordering, and more.

The core app communicates with a large relational database.

The core exposes functionality via an HTML interface.



Monoliths: Pros

- build
- test
- deploy

But then you realize not all is bad.

Monoliths offer some distinct advantages.

For example, they're straightforward to...

Build: you can hit build in your IDE and it just compiles and runs

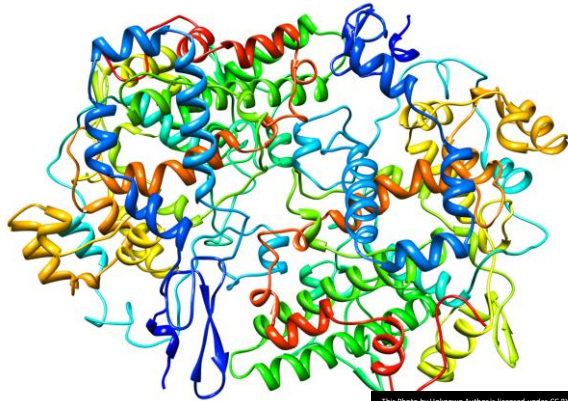
Test: for instance, you can click add to cart and test if it adds to the cart

Deploy: sitting at your desk, you can right-click deploy straight to production

Many successful apps that exist today were created as monoliths.

Monoliths: Cons

- Redeploy entire app
- Changes are tough
- Complicated
- Lack of stability
- Entropy
- Bad on résumé



Each release as small as it may be requires a full deployment of the entire application. (small db change in basket module requires db and app deploy)

- You fear making changes - each change has unintended and costly side effects.
- The app has become so overwhelmingly complicated that no single person understands it.

New features/fixes become tricky, time-consuming, and expensive to implement.

- One unstable component can crash the entire system.
- Architectural erosion sets in as the code base deteriorates with never-ending "special cases."

The consultants tell you to rewrite it.

- New technologies and frameworks aren't an option. No Kubernetes for instance.
- Bad on resume

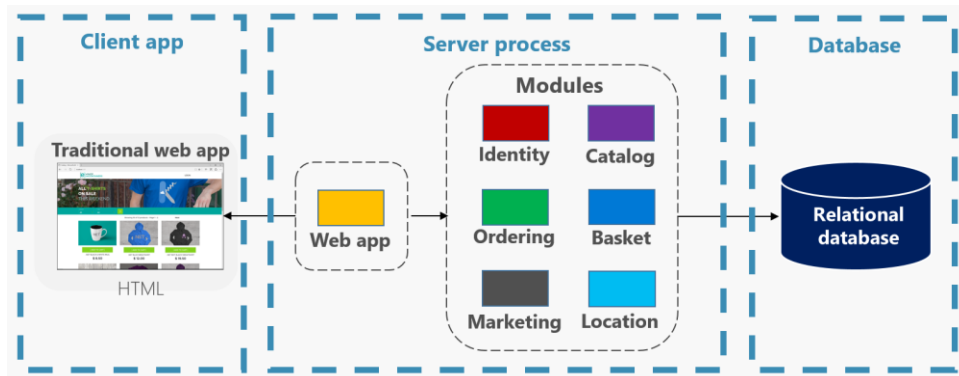
At some point, however, you begin to feel uncomfortable.

You find yourself losing control of the application.

As time goes on, the feeling becomes more intense and you eventually enter a state known as the **Fear Cycle**.

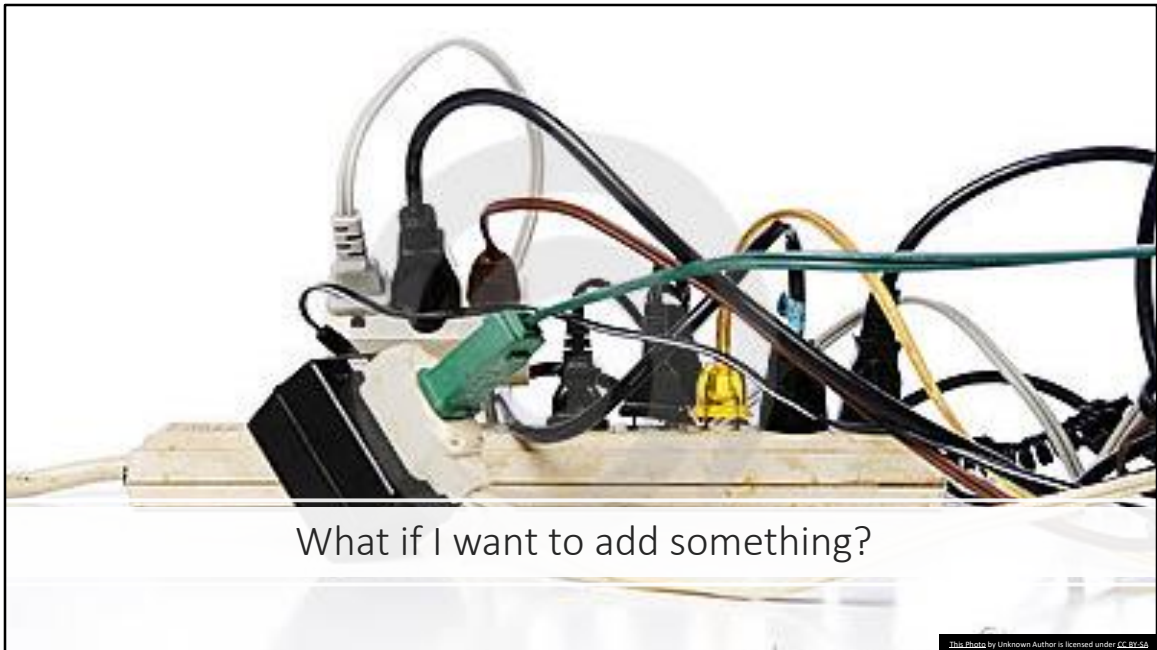
Part 2: Monolith to Microservices – an unexpected journey

Monoliths to Microservices: An unexpected journey



Credit: [Microsoft.com](https://microsoft.com) Introduction to cloud-native applications

I totally copied that unexpected journey title from the hobbit movie
So at this point, you realize there is no option but to start splitting up your system.
You use some domain design principles and split your web app into modules
Each module is self-contained and encapsulates its own code, data, and dependencies
You have the identity module that handles authentication and authorization
You have catalog that shows and maintains inventory
You have basket that handles the cart
You have ordering that handles payments



What if I want to add something?

Congrats that's the bare minimum

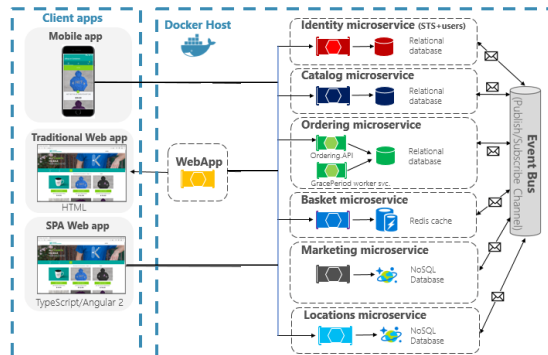
At this point all you can do is deploy and troubleshoot separately

Only apps

Database is still a monolith

What happens when you add a mobile app?

Bring on the Containers!!!



Credit: [Microsoft.com](https://microsoft.com) Introduction to cloud-native application

Then you do more research and discover the world of containers
Now each module is deployed in a software container
and managed by a container orchestrator like Kubernetes.
Instead of a large relational database,
each service owns its own datastore,
the type of which varies based upon the data needs.
Note how some services depend on a relational database, but others on NoSQL
databases.
One service stores its state in a distributed cache.

Challenges in a system of containers

Problem scenarios:

- Signatures
- Formats
- Security
- Networking
- Performance

Then being the pragmatic developer that you are,
You realize that containers alone are not the solution to every problem in the world

Problem scenarios:

- Signatures, multiple microservices have to change signatures
- Formats, multiple microservices have to deal with multiple service formats
- Each backend API may have it's own system of security
- Each microservice would have to implement front end security
- Networks, some on-premises, some cloud private, some public cloud
- Performance, some may get more intensive hits



These problems keep you awake at night.

One day you're at a conference and end up talking to a random wise person

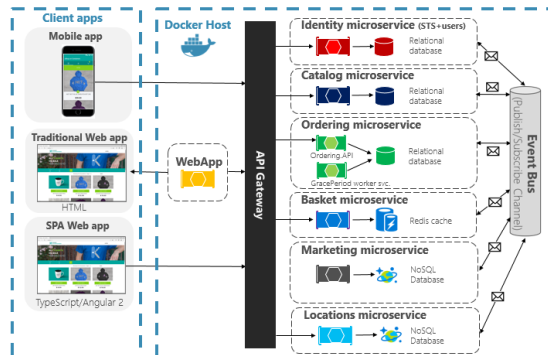
The bald guy pulls a Morpheus and says:

What if I told you there is something that will

- handles multiple microservices including change management for signatures
- handles multiple service formats – REST, JSON, Swagger, SOAP
- Allow Each backend API to keep their own system of security
- Allow each front end to bring their own security
- allow some on-premises, some cloud private, some public cloud
- Handle intensive hits on certain components only

Part 3: API Gateways

API Gateway



Credit: [Microsoft.com](https://microsoft.com) Introduction to cloud-native applications

how all traffic routes through an API Gateway service that is responsible for routing traffic to the core back-end services

- Enforces many cross-cutting concerns like
- Signatures
- Formats
- Security
- Networking
- Performance

Most importantly, the application takes full advantage of the scalability and resiliency features found in modern cloud platforms.

API Gateway Vendors



AZURE API
MANAGEMENT



APIGEE



AWS GATEWAY



EXPRESS GATEWAY

If you research API Gateways and discover you have multiple options

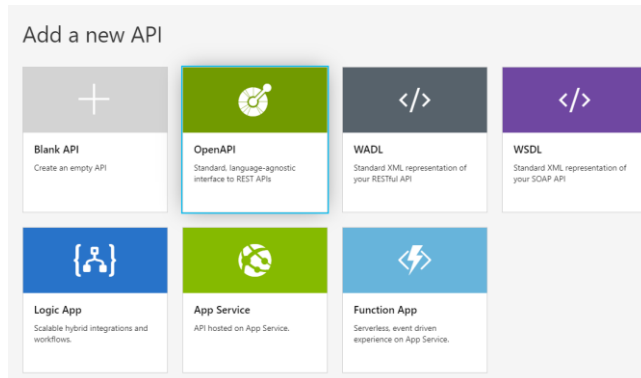
Azure API Management – Multi-node replication, PaaS, proprietary, In-built caching, On-premises connect via VPN

APIGee – Nodes with different roles, proprietary, needs separate datastore for caching, can deploy on-premises and Google Cloud

AWS Gateway - Multi-node replication, PaaS, proprietary, In-built caching, On-premises connect via VPN

Express Gateway – Multi-node replication, OSS, needs separate datastore for caching, can deploy on-premises

JSON, SOAP, REST, Swagger – these are a few of my favorite things



You discover that most of these API Gateway providers handle multiple formats

REST API (blank api) – add each operation

Open API (swagger) – automatic


WSDL and WADL – discovery languages for REST and SOAP respectively

Swagger is good

Create from OpenAPI specification

Basic | Full

OpenAPI specification



or

(maximum size 4 MB)

Display name

Name

API URL suffix

Base URL

REVISION 1 UPDATED Jan 9, 2020, 10:40:51 AM

Design Settings Test Revisions Change log

All operations

POST	Add a new pet to the store	...
POST	Create user	...
POST	Creates list of users with given input array	...
POST	Creates list of users with given input array	...
DEL	Delete purchase order by ID	...
DEL	Delete user	...
DEL	Deletes a pet	...
GET	Find pet by ID	...
GET	Find purchase order by ID	...

Automatically adds
Machine readable
Conforms to original API intent

Signatures, revisions and versions

Create a new API as a version of Swagger Petstore (current revision)

Versioning creates a new API, which is linked to an existing API through versioning scheme.

• Name
version2

• Versioning scheme
Path

• Version Identifier
v2

Usage example
/v2

Products
No products selected

• To publish the API, you must associate it with a product. [Learn more.](#)

Create Cancel

REVISION 2 UPDATED Jan 9, 2020, 10:58:23 AM

Revision 2 - Updated Jan 9, 2020, 10:58:23 AM



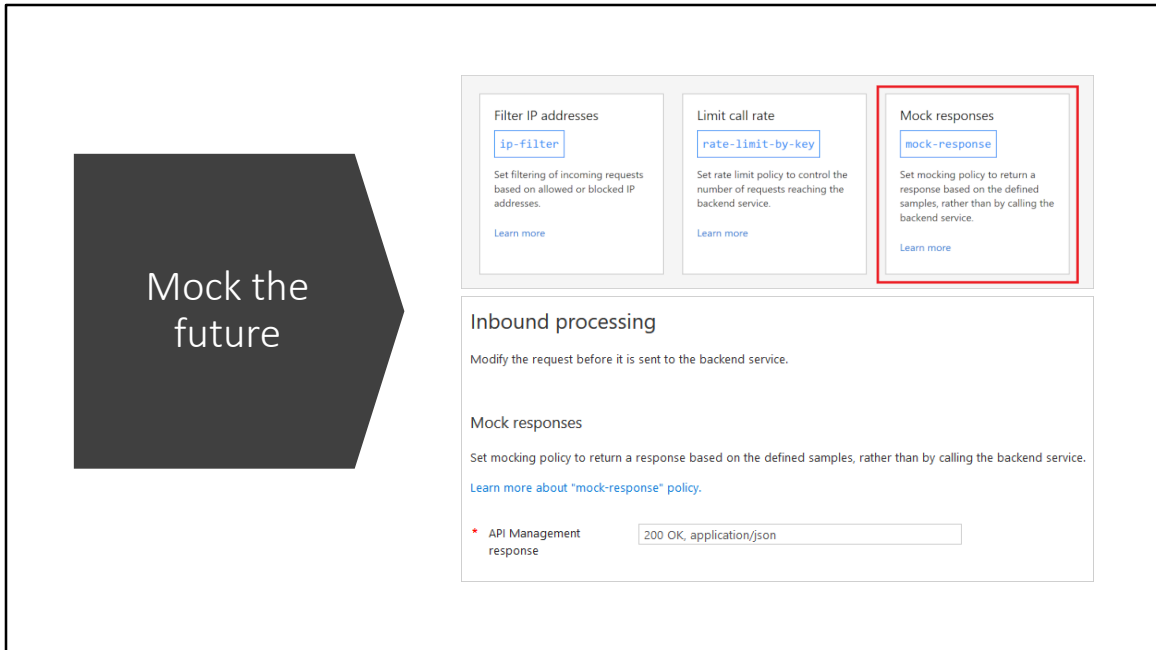
Revision 1 - Updated Jan 9, 2020, 10:40:51 AM



When your API is ready to go and starts to be used by developers, you eventually need to make changes to that API at the same time not disrupt callers of your API. It's also useful to let developers know about the changes you made. We can do this using **revisions**

There are times when it is impractical to have all callers to your API use exactly the same version.

When callers want to upgrade to a later version, they want to be able to do this using an easy to understand approach. It is possible to do this using **versions** in Azure API Management.



Let's say your mobile team wants a brand new API call that will not impact the web team

You want to create and provision this separately

Additionally your backend team won't start work at least until next sprint

How can you get your mobile team being productive and start coding against this API?

Your developers are your customers

+ Add Columns Refresh ?

Products let you group APIs, define terms of use, and runtime policies. API consumers can subscribe to a product on the developer portal to obtain a key to call your APIs. [Learn more](#)

Search to filter items...

Display name	Access control	State	
Starter	Administrators, Developers, Guests	Published	...
Unlimited	Administrators, Developers, Guests	Published	...

When you have front end developers using your APIs, your developers are your consumers and your APIs are your products a product contains one or more APIs as well as a usage quota and the terms of use. Once a product is published, developers can subscribe to the product and begin to use the product's APIs.

Your developers are your subscribers

[+ Add subscription](#) [Columns](#) [Refresh](#) | [?](#)

API consumers can subscribe to Products to start using your APIs. [Learn more](#) ×

State All Pending approval Scope All ▼

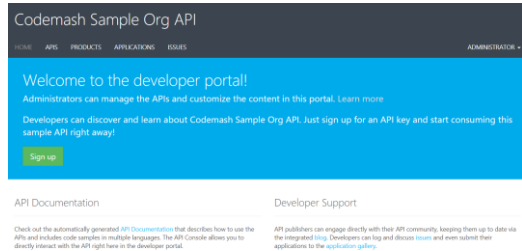
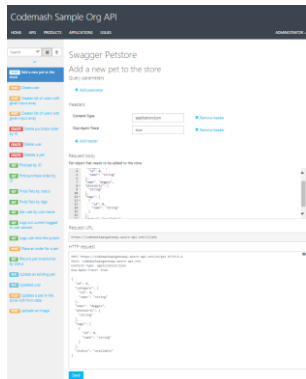
Display name	Primary key	Secondary key	Scope	State	Owner	Allow tracing	
	<div>c933205d7... </div>	<div>2d324f79dc... </div>	Product: Starter	Active	Administrator	✓	...
	<div>*****...</div>	<div>*****...</div>	Product: Unlimited	Active	Administrator	✓	...
Built-in all-access ...	<div>*****...</div>	<div>*****...</div>	Service	Active		✓	...

When you publish APIs through API Gateway, it's easy and common to secure access to those APIs by using subscription keys. Developers who need to consume the published APIs must include a valid subscription key in HTTP requests when they make calls to those APIs.

Otherwise, the calls are rejected immediately by the API Management gateway. They aren't forwarded to the back-end services.

Rotate keys with good DevOps practices

Provide developers with a playground



```
<rate-limit-by-key calls="10"  
    renewal-period="60"  
    counter-key="@context.Request.IpAddress" />
```

```
<quota-by-key calls="1000000"  
    bandwidth="10000"  
    renewal-period="2629800"  
    counter-key="@context.Request.IpAddress" />
```

```
<rate-limit-by-key calls="100"  
    renewal-period="60"  
    counter-key="@request.Headers.GetValueOrDefault("Rate-Key","")"/>
```

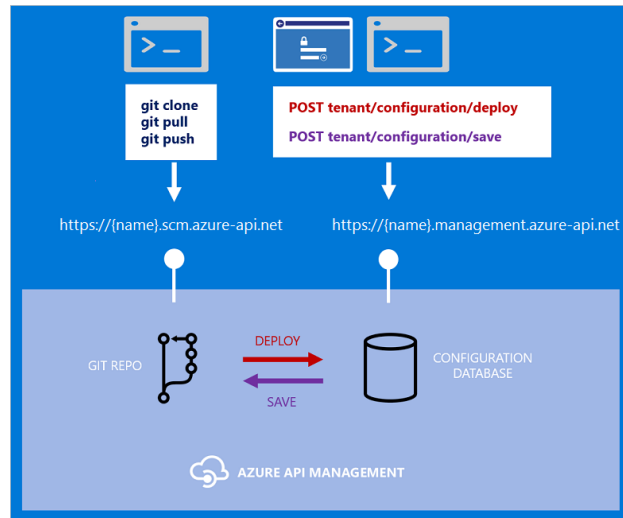
Keep rouge devs and infinite loops in check

The following policies restrict a single client IP address to only 10 calls every minute , with a total of 1,000,000 calls and 10,000 kilobytes of bandwidth per month.
Based on IP address

This enables the developer's client application to choose how they want to create the rate limiting key.

The client developers could create their own rate tiers by allocating sets of keys to users and rotating the key usage.

DevOps all the things



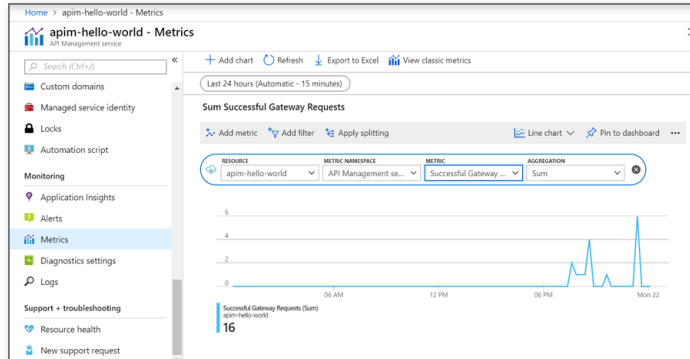
<https://docs.microsoft.com/en-us/azure/api-management/api-management-configuration-repository-git>

Configuration versioning - download and store different versions of your service configuration

Bulk configuration changes - make changes to multiple parts of your service configuration in your local repository and integrate the changes back to the server with a single operation

Familiar Git toolchain and workflow - use the Git tooling and workflows that you are already familiar with

Monitor and measure



Your API gateway providers should help you visualize, query, route, archive, and take actions on the metrics or logs

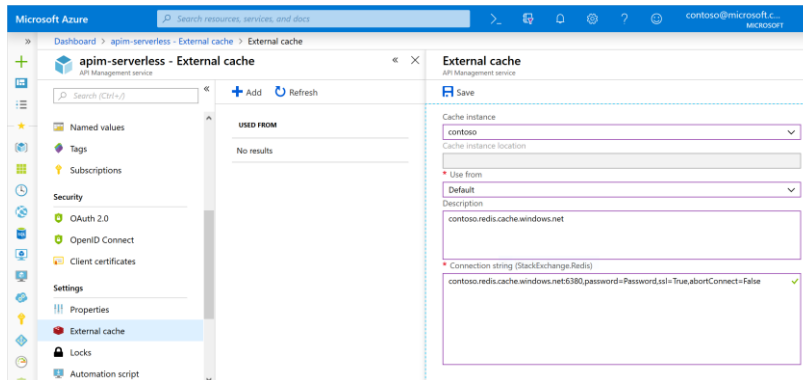
View activity logs

View diagnostic logs

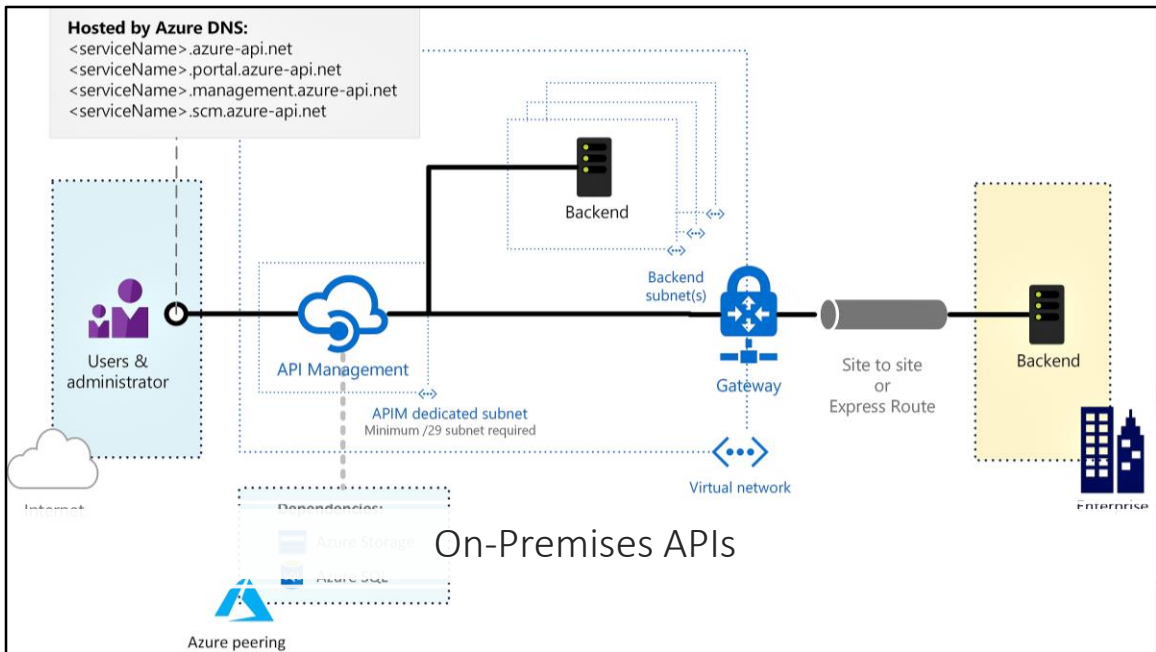
View metrics of your API

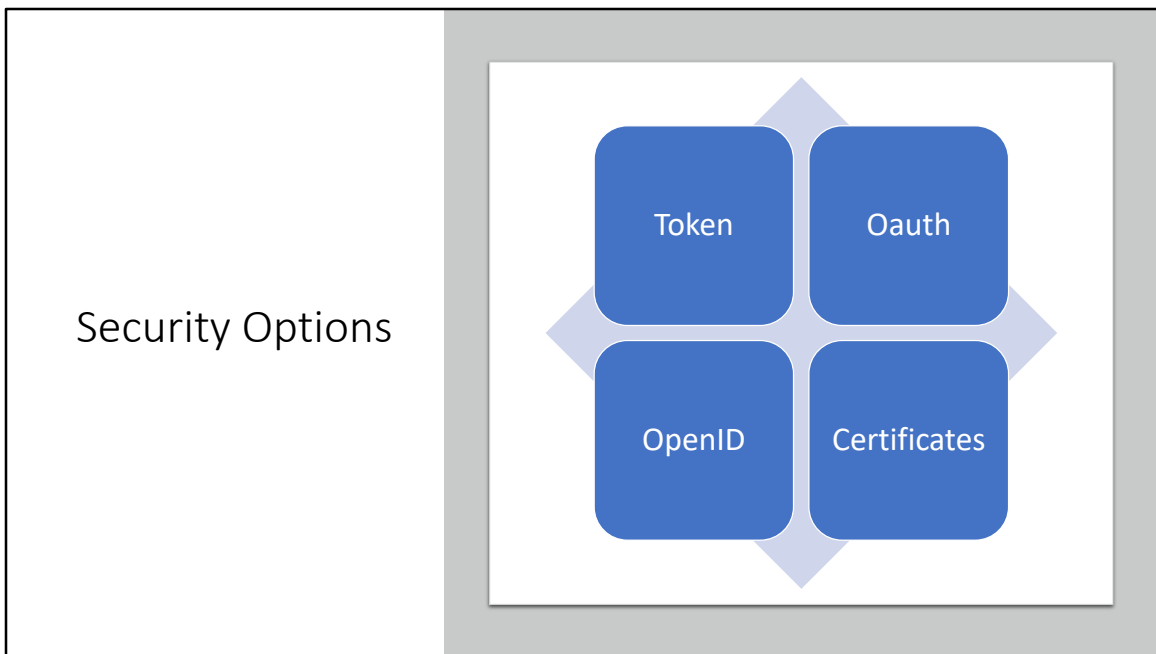
Set up an alert rule when your API gets unauthorized calls

Cache is money



In-built and external





oath and open id

These are important for you in the scenario that you start working another big chain that handles pet food

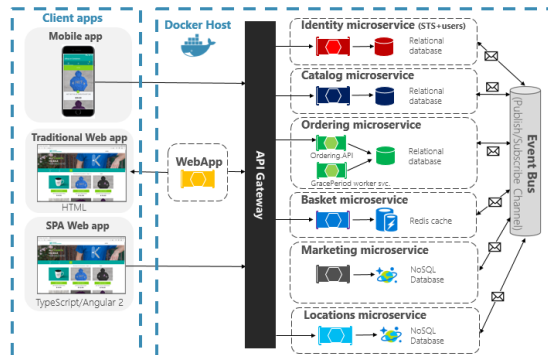
allows you to register an external client and authorize it without having to trust their client.

If you wanted to allow a B2B service at Home Depot see your inventory you could allow them to register the integration and limit access to just inventory.

You could allow a client to integrate your services into their applications with rate limiting or as a licensed api.

If there was a breach or a bad actor, you can revoke the api access to that client.

API Gateway



Credit: [Microsoft.com](https://microsoft.com) Introduction to cloud-native applications

how all traffic routes through an API Gateway service that is responsible for routing traffic to the core back-end services

- Enforces many cross-cutting concerns like
- Signatures
- Formats
- Security
- Networking
- Performance

Most importantly, the application takes full advantage of the scalability and resiliency features found in modern cloud platforms.

Resources

- My Blog Post: <https://www.nebbiatech.com/2019/05/30/azure-apim-and-microservices-2-peas-in-a-pod-container/>
- Azure API Management: <https://docs.microsoft.com/en-us/azure/api-management/>
- Introduction to cloud native architecture: <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduction>

Santosh Hari

Azure Consultant @ New Signature


Azure MVP

President, Orlando .NET UG

Organizer, Orlando Codecamp

 santoshhari.wordpress.com

 santosh.hari@newsignature.com

 @_s_hari

 /in/santoshhari



NEWSIGNATURE