Welcome to the Singapore

2019

Global Azure

BOOTCAMP

Resilient Microservices with AKS

Nilesh Gule

@nileshgule | www.HandsOnArchitect.com

A BIG thank you to the 2019 Global Sponsors!

Microsoft

kemp

JET BRAINS

Cloudionix

Progate

Skill Me UP
Powered by opsgility

SERVICEBUS 360

2019
Global Azure
BOOTCAMP

Agenda

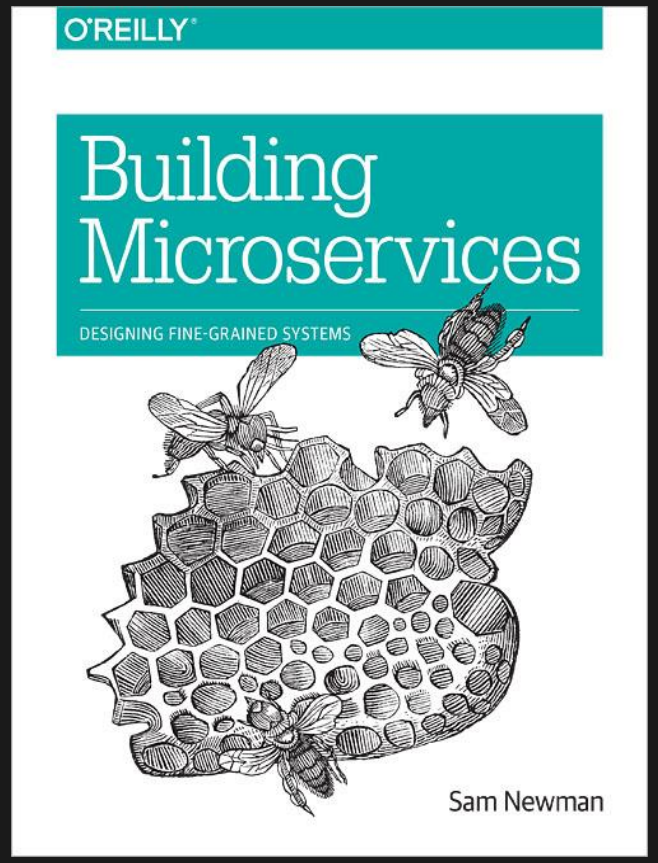
- Microservices Principles
- Microservices Architecture Patterns
- Service Mesh Overview
- Resiliency Patterns

Microservice

Autonomous services that work together

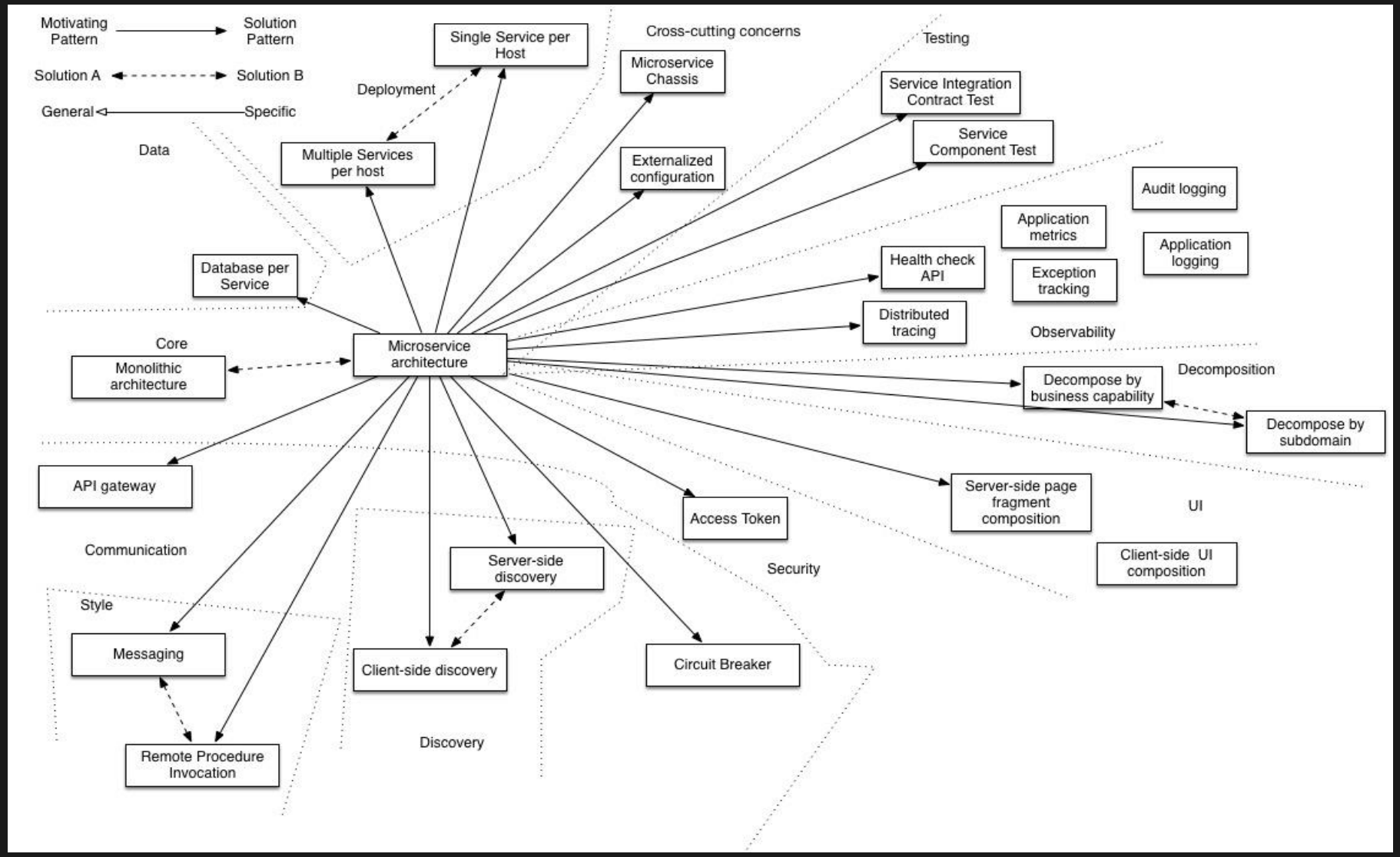
Principles of Microservices

- Modelled Around Business Domain
- Culture of Automation
- Hide Implementation Details
- Decentralize All Things
- Deploy Independently
- Customer First
- Isolate Failure
- Highly Observable



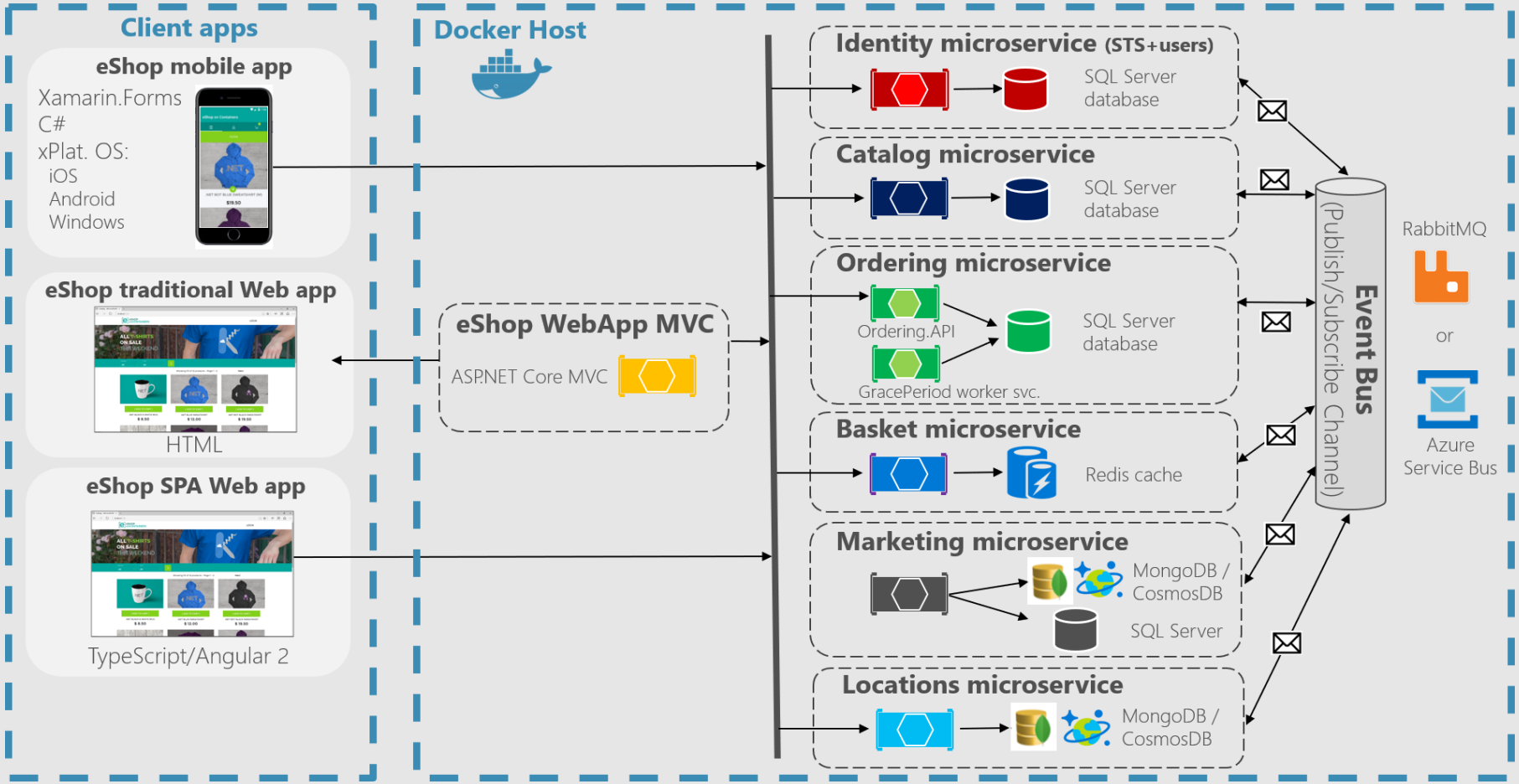
https://samnewman.io/talks/principles-of-microservices/

Microservices Architecture Patterns



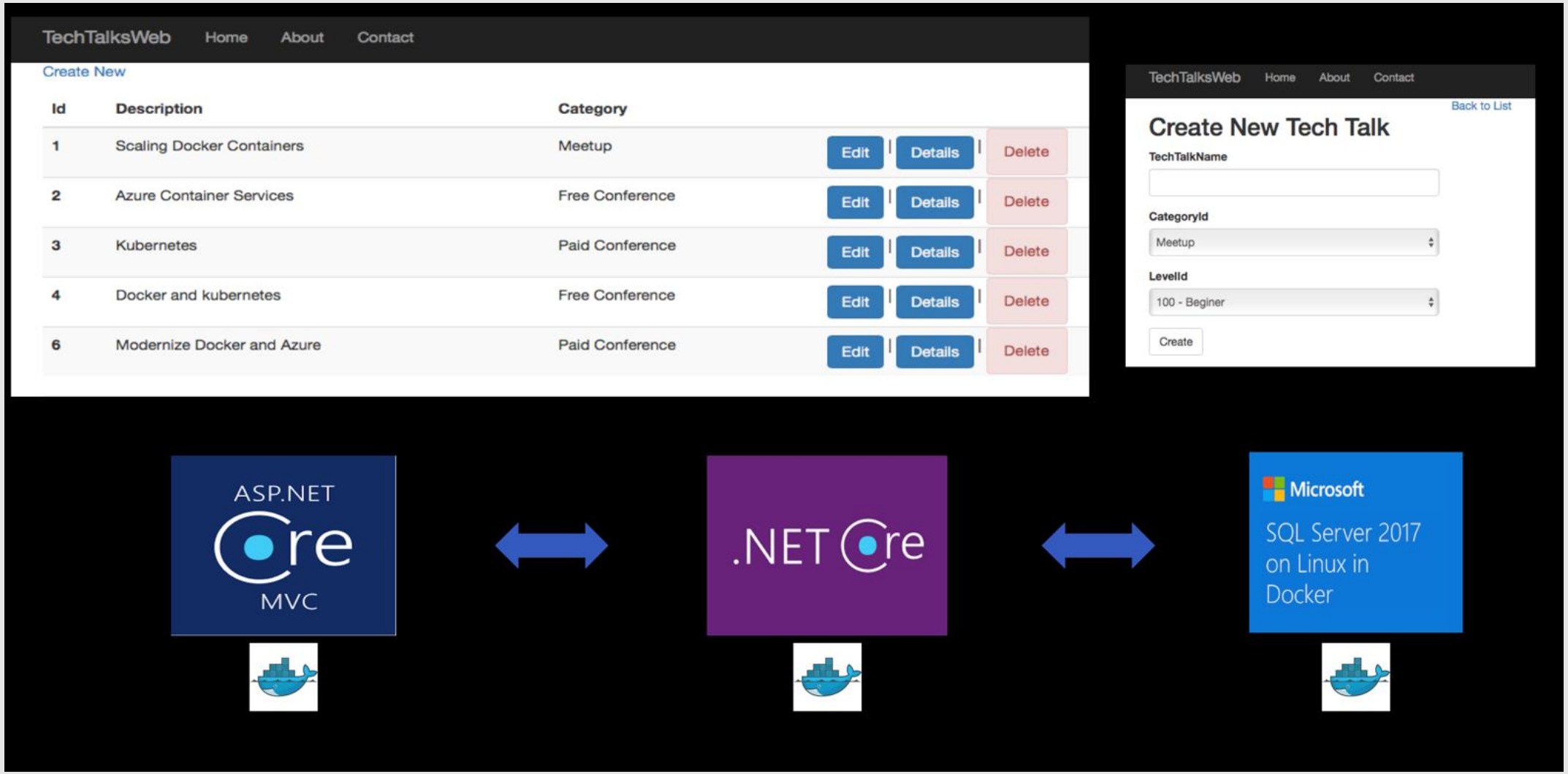
https://microservices.io/patterns/microservices.html

Reference Application - eShopOnContainers



https://github.com/dotnet-architecture/eShopOnContainers

Reference Application - TechTalks



https://github.com/NileshGule/AKS-learning-series

eShop & TechTalks Demo

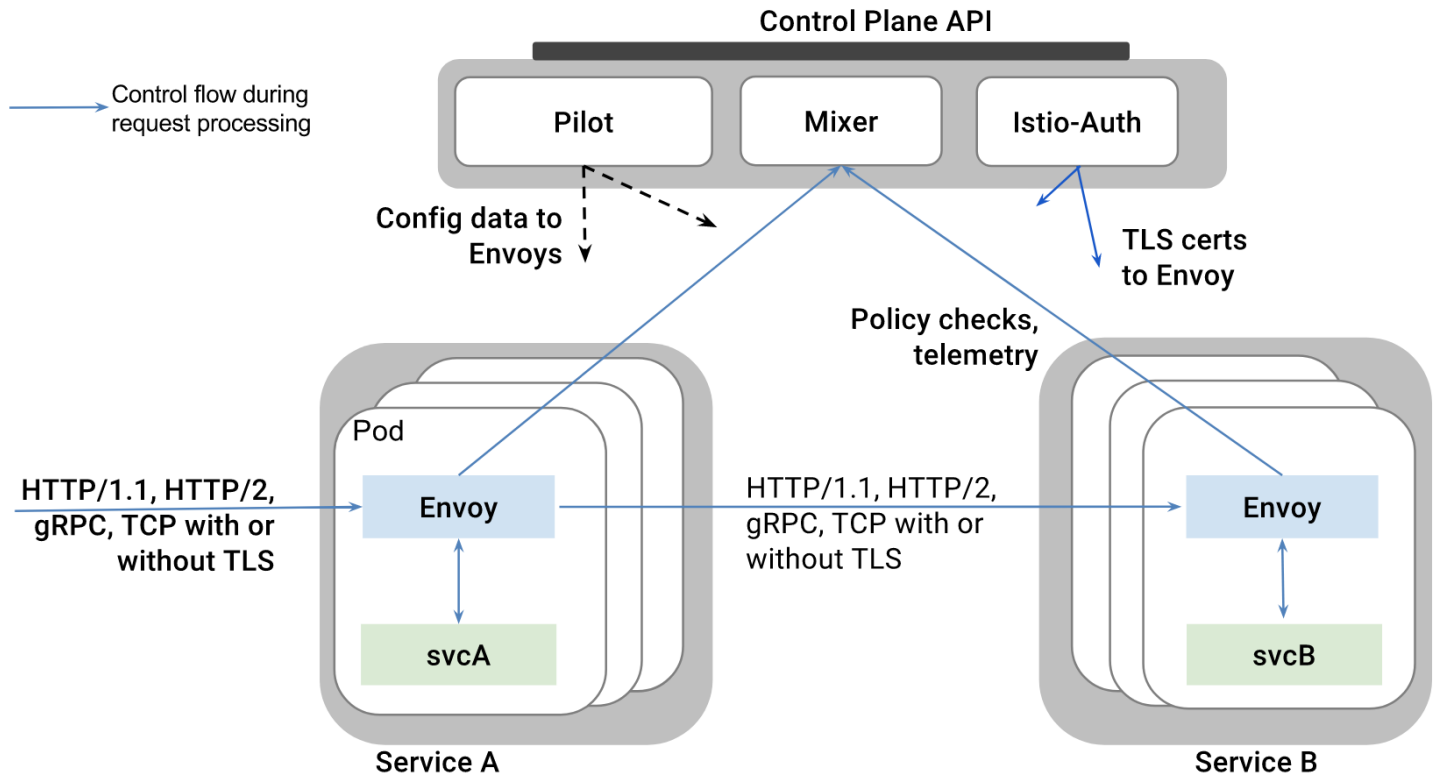
Service Mesh

Software infrastructure layer for controlling and monitoring internal, service-to-service traffic in Microservice application

Service Mesh

Need for Service Mesh

- Traffic Monitoring
- Access Controls
- Discovery
- Security
- Resiliency



Istio Architecture

Istio Features – No code changes

- Automatic load balancing for
  - HTTP
  - gRPC
  - WebSocket
  - TCP traffic
- Fine grain control of traffic behavior with
  - Rich Routing Rules
  - Retries
  - failover and fault injection
- Pluggable policy layer and Configuration API supporting access controls, rate limits and quotas
- Automatic metrics, logs, and trace for all traffic within cluster
- Secure service-to-service communication

Istio Sidecar Demo

Resiliency

Ability of system to Recover from failure and continue to function

Microservices Resiliency Patterns

- Retry
- Timeout
- Fallback
- Bulkhead Isolation
- Circuit Breaker
- Leader Election
- Compensating Transactions
- Queue Based Load Levelling
- Scheduler Agent Supervisor
- Health Endpoint Monitoring

Retry

Enable application to handle transient failures

Retry

Enable application to handle transient failures

Context

Transient fault occurs due to loss of network connectivity causing temporary unavailability of service or timeout

Solution

Handle failures using one of the strategies

- Cancel
- Retry
- Retry After Delay (Progressive Backoff)

https://docs.microsoft.com/en-sg/azure/architecture/patterns/retry

Retry

.Net example Progressive Backoff - Polly

```
static IAsyncPolicy<HttpResponseMessage> GetRetryPolicy()
        {
            return HttpPolicyExtensions
              .HandleTransientHttpError()
              .OrResult(msg => msg.StatusCode == System.Net.HttpStatusCode.NotFound)
              .WaitAndRetryAsync(6, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2,
retryAttempt)));

        }

//Add retry policy to service
services.AddHttpClient<ICampaignService, CampaignService>()

.AddHttpMessageHandler<HttpClientAuthorizationDelegatingHandler>()
                .AddPolicyHandler(GetRetryPolicy());
```

Retry

Service Mesh example - Istio

```yaml
metadata:
  name: my-rule
  namespace: default
spec:
  destination:
    name: ratings
  route:
  - labels:
      version: v1
  httpReqRetries:
    simpleRetry:
      attempts: 3
      perTryTimeout: 2s
```

http://goupaz.com/docs/reference/config/traffic-rules/routing-rules.html#httpretry

Timeout

Don’t wait forever

Timeout

Beyond a certain wait, a success result is unlikely

Context

Distributed services can take long time to respond

Solution

Terminate operation after specified amount of time

Timeout

.Net example Optimistic Timeout - Polly

```
CancellationTokenSource userCancellationSource = new
CancellationTokenSource(); // hooked up to 'cancel' button

Policy timeoutPolicy = Policy.TimeoutAsync(30, TimeoutStrategy.Optimistic);

HttpResponseMessage httpResponse = await timeoutPolicy
    .ExecuteAsync(
        async ct => await httpClient.GetAsync(requestEndpoint, ct),
        userCancellationSource.Token
        );
```

Timeout

Service Mesh example - Istio

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
    timeout: 0.5s
```

https://istio.io/docs/tasks/traffic-management/request-timeouts/

Fallback

Provide graceful degradation in event of failure

Fallback

Provide substitute value in event of failure

Context

Certain functionality can fail, instead of returning failure result, provide default functionality or downgraded features

Solution

Plan for alternate graceful degradation when outright failures occurs

Cached / default results

Bulkhead Isolation

Don’t let one fault sink the whole ship

Bulkhead Isolation

Avoid faults in one part of the system to take entire system down

Context

Multiple services have multiple consumers. Excessive load or failure in one service impacts all consumers of the service

Solution

Partition service instances into different groups, based on consumer load and availability requirements. Isolate failures and sustain service functionality for some consumers even during failure

https://docs.microsoft.com/en-sg/azure/architecture/patterns/bulkhead

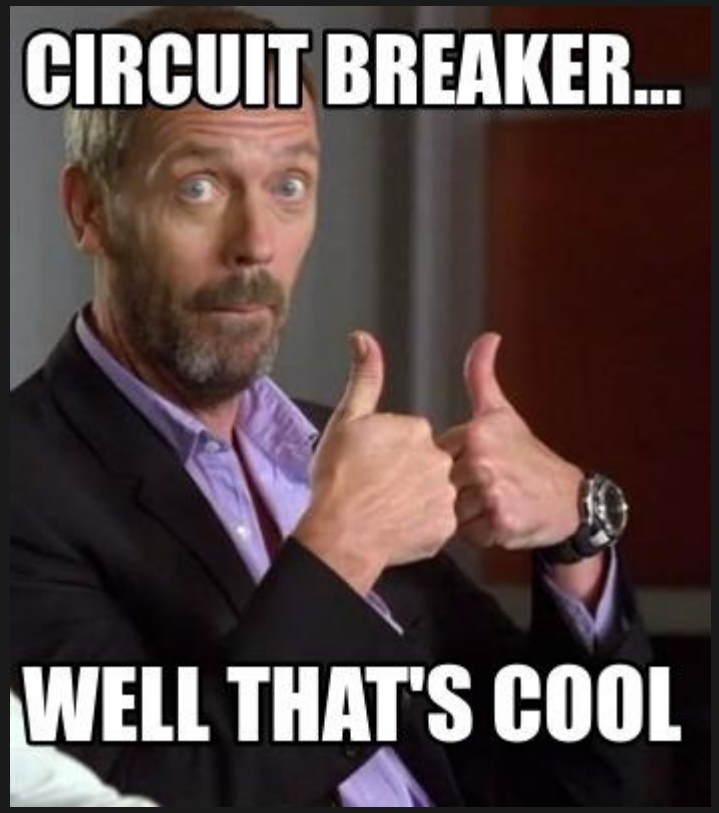
Bulkhead Isolation

Limit CPU and Memory

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: techtalksweb-v2
spec:
  containers:
  - name: techtalksweb
    image: nileshgule/techtalksweb:v2
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "1“
```

Circuit Breaker

Prevent repeated retries to an operation that is likely to fail



Circuit Breaker

Prevent repeated retries to an operation that is likely to fail

Context

Faults due to unanticipated events and take much longer to fix.

Failures in one service causes cascading failures.

Solution

Monitor the number of recent failures using Circuit Breaker proxy for operations that might fail. Use the information to allow operation or return exception immediately.

https://docs.microsoft.com/en-sg/azure/architecture/patterns/circuit-breaker

Circuit Breaker

.Net example - Polly


```
static IAsyncPolicy<HttpResponseMessage> GetCircuitBreakerPolicy()
{
    return HttpPolicyExtensions
        .HandleTransientHttpError()
        .CircuitBreakerAsync(5, TimeSpan.FromSeconds(30));
}

services.AddHttpClient<IOrderingService, OrderingService>()
    .AddHttpMessageHandler<HttpClientAuthorizationDelegatingHandler>()
    .AddHttpMessageHandler<HttpClientRequestIdDelegatingHandler>()
    .AddPolicyHandler(GetRetryPolicy())
    .AddPolicyHandler(GetCircuitBreakerPolicy());
```

Circuit Breaker

Java example - Hystrix


```
@SpringBootApplication
@EnableHystrixDashboard
@EnableCircuitBreaker
public class SpringHystrixSchoolServiceApplication {

    public static void main(String[] args) {

SpringApplication.run(SpringHystrixSchoolServiceApplication.class, args);
    }
}
```

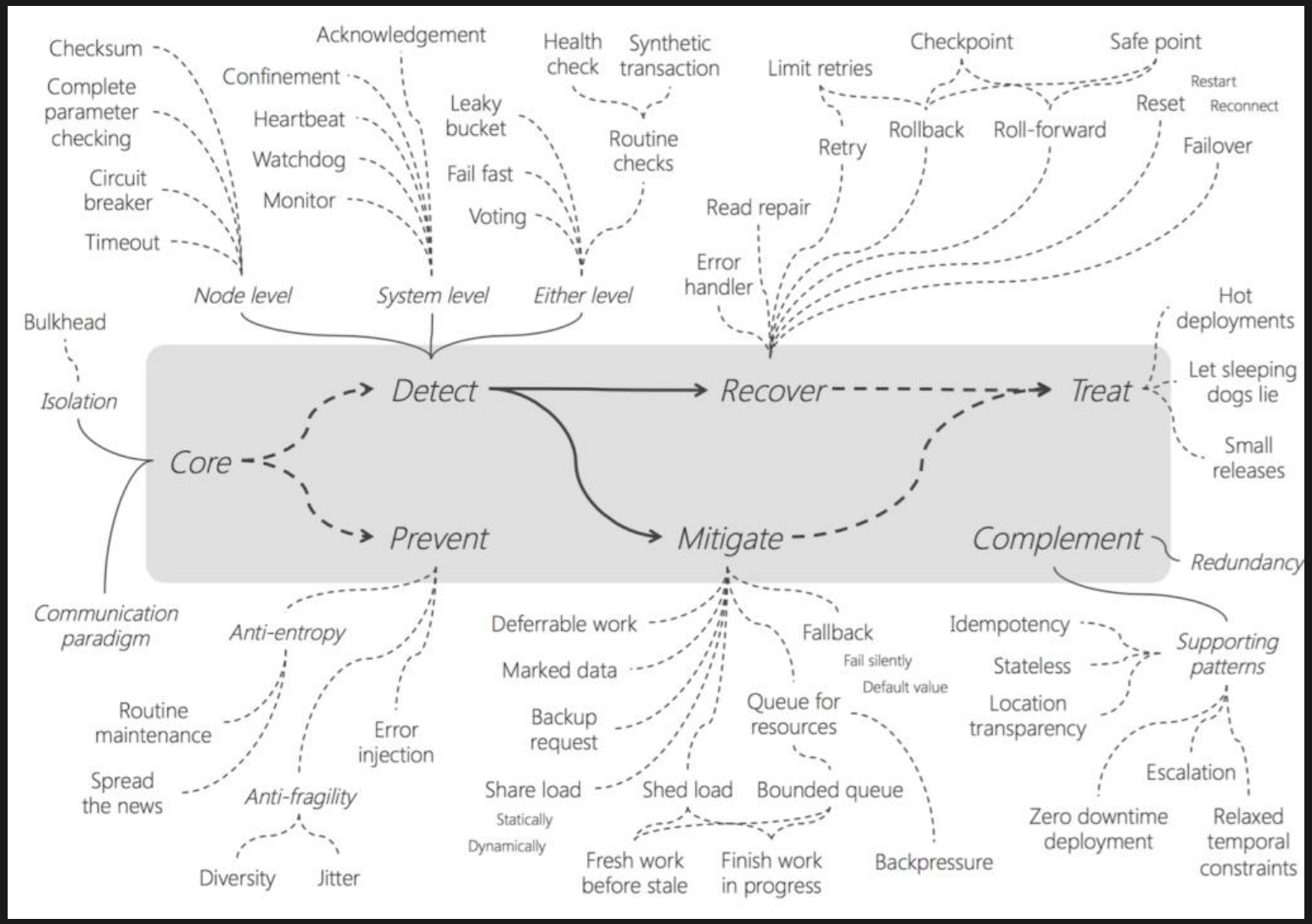
Circuit Breaker

Service Mesh example - Istio

```yaml
metadata:
  name: reviews-cb-policy
  namespace: default
spec:
  destination:
    name: reviews
    labels:
      version: v1
  circuitBreaker:
    simpleCb:
      maxConnections: 100
      httpMaxRequests: 1000
      httpMaxRequestsPerConnection: 10
      httpConsecutiveErrors: 7
      sleepWindow: 15m
      httpDetectionInterval: 5m
```

Istio Cirtcuit Breaker

Resiliency Patterns



https://developers.redhat.com/blog/2017/05/16/it-takes-more-than-a-circuit-breaker-to-create-a-resilient-application/

Key Takeaways

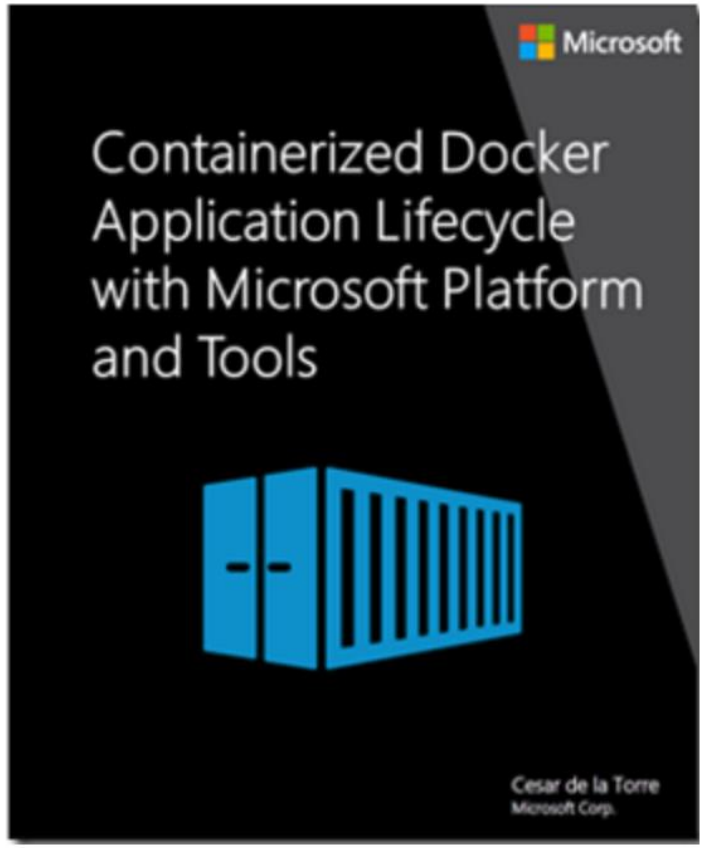
Embrace failure

- Services should fail separately
- Stop cascading of failures
- Fail fast and independently and rapidly recover
- Degrade gracefully where possible

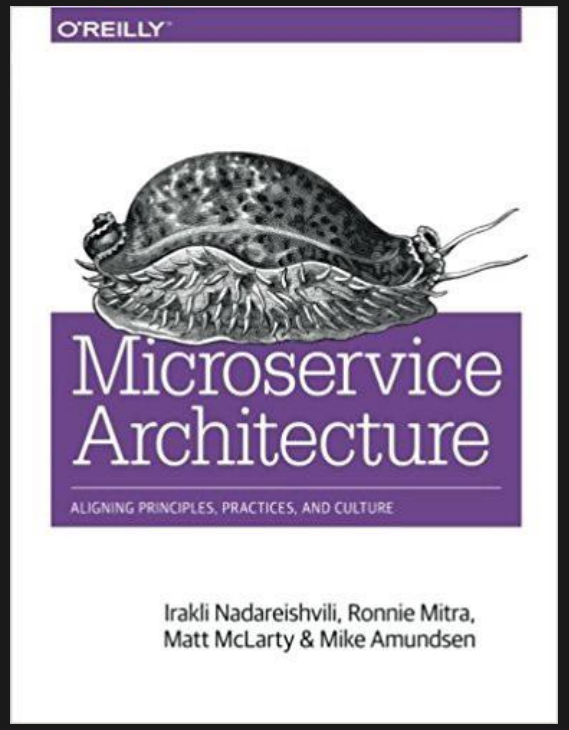
References

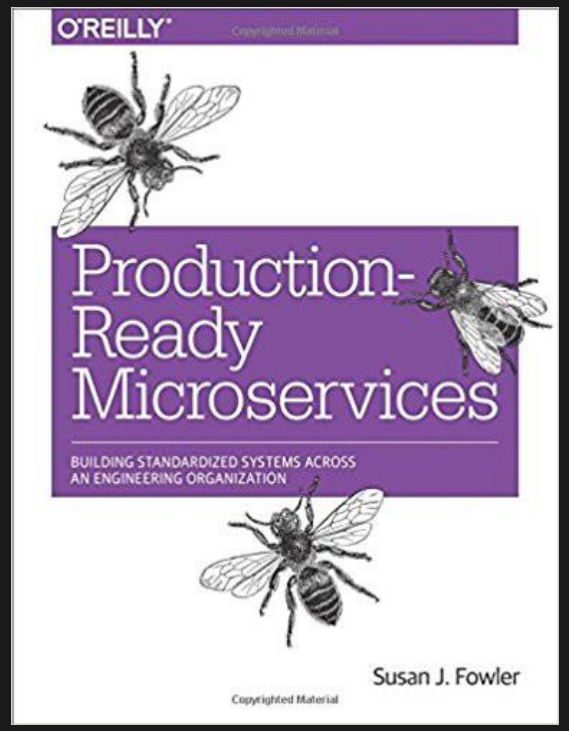
- eShopOnContainers
- Resiliency Patterns
- 8 fallacies of Distributed Computing
- 12 factor apps
- Polly
- Istio
- Istio Service Mesh
- Hystrix patterns
- History of Service Mesh
- Rise of Service Mesh Architecture

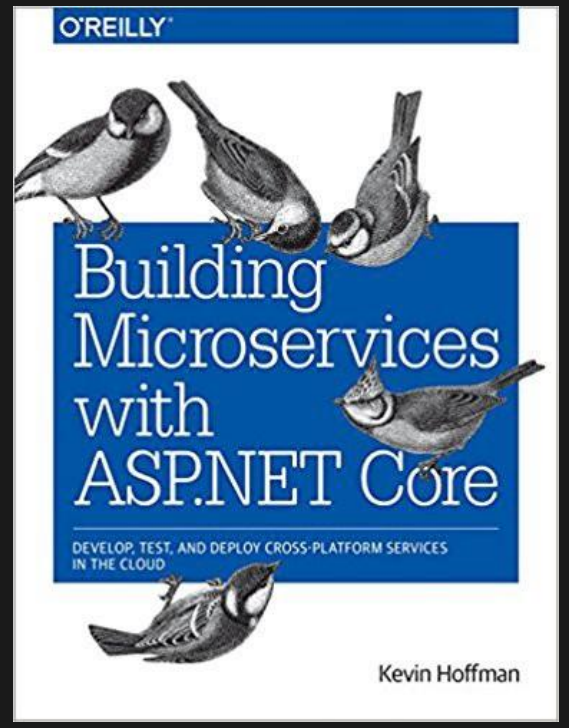
References

Architecting & Developing	Containers Lifecycle & CI/CD	App patterns with Xamarin.Forms
		

References


O'REILLY
Microservice Architecture
ALIGNING PRINCIPLES, PRACTICES, AND CULTURE
Irakli Nadareishvili, Ronnie Mitra, Matt McLarty & Mike Amundsen


O'REILLY
Production-Ready Microservices
BUILDING STANDARDIZED SYSTEMS ACROSS AN ENGINEERING ORGANIZATION
Susan J. Fowler
Copyrighted Material


O'REILLY
Building Microservices with ASP.NET Core
DEVELOP, TEST, AND DEPLOY CROSS-PLATFORM SERVICES IN THE CLOUD
Kevin Hoffman


O'REILLY
Introducing Istio Service Mesh for Microservices
Build and Deploy Resilient, Fault-Tolerant Cloud-Native Applications
Christian Posta & Burr Sutter

Thank you very much

Code with Passion and Strive for Excellence

https://github.com/NileshGule/ABC2019

https://www.slideshare.net/nileshgule/presentations
https://speakerdeck.com/nileshgule/

$whoami

{

"name" : "Nilesh Gule",
"website" : "https://www.HandsOnArchitect.com",
"github" : "https://github.com/NileshGule"
"twitter" : "@nileshgule",
"linkedin" : "https://www.linkedin.com/in/nileshgule",
"email" : "nileshgule@gmail.com",
"likes" : "Technical Evangelism, Cricket"

}

Microsoft® Most Valuable Professional


Q&A