# Archetypal Secure Application Design Patterns: The Next Evolution -Or- Layered Pattern Stacks as Code (LPSaC)

BY JOE GERBER

HISTORY@FRII.COM

1/20/2021

# Special Thanks to Absent Friends:

**Jay Reynolds**
(---)

*Application Security Program Lead*
*CISSP*

**Chris Wells, RIP**
(---)

*Application Security Architect*
*CISSP*

**Christian Price**
(@DevSecOpsXian)

*Cloud Security Architect*
*CISSP, CISM, CISA*

## Of Note:

*The content of this presentation represents a synthesis of our collected experience and opinions, informed by the experience and opinions of the many humans whom have influenced our journey. To all these humans and experiences we are grateful.*

*To be clear, this presentation does not represent any of our employers, past or present, and we are grateful to our employers' support for our independent community contributions such as this.*

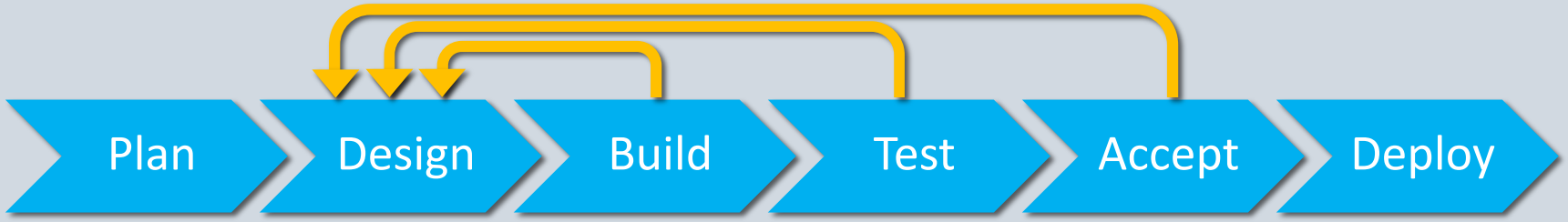
# The Problem: how to effectively << *shift left*

How many times have you seen something during a security evaluation that makes you shake your head?

Do you find yourself saying “if only they had involved us sooner…”?

---

But what does it really mean to shift left?




The current trend is towards earlier integration of better security testing during CI/CD → earlier feedback is better.

We constantly want to be engaged at the design phase, but security teams can’t scale and become a bottleneck

**We are missing a huge opportunity to influence design by speaking the language of patterns**

# Goals / Objectives

Amplify software architecture & design as a critical element of AppSec

Communicate relevance & importance of patterns in context of microservices

Articulate some modern design principles

Illustrate an approach to building a pattern catalog

Help others on the journey

Inspire engagement and contribution

Using patterns to shift security left

*This is a journey, We are by no-means 'done' with this topic following this talk.*
*This is an area of continuing passion, inquiry, research, and advocacy for us.*

# Our Iterative Approach:

1. Propose a set of software architecture and software design patterns at various levels of detail
2. Subject those patterns to rigorous analysis, including:
   - Threat Modeling
   - Attack Map / Analysis
   - Live attack trial implementation (RedTeam, Pen Test, etc. – pick your favorite terminology for an intelligent unbounded attacker)
   - Other analysis approaches we may not have thought of here
3. Learn from the results
4. Goto (1)

# Scope of this presentation:

Survey / Review common architecture patterns:

- applications/software
- infrastructure/deployment

Show how they are broadly applicable

- One interesting test:  can the patterns secure some of the riskiest apps?

Show the world as it looks to software and software creators:

- The context in which the software exists
- The other systems with which the software interacts, and the AppSec responsibilities of each
- The components of the software, and the AppSec responsibilities of each
- How to meet those responsibilities

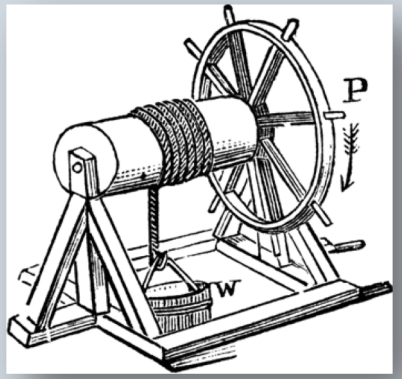
# Patterns

Provide reusable solutions to common problems

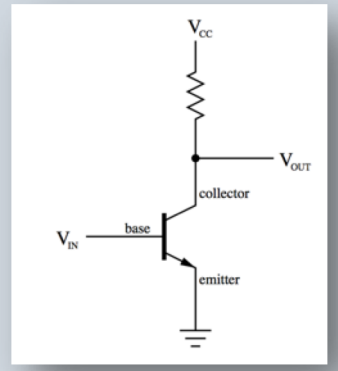
Provide a consistent language to communicate about solution composition

Can be assessed for weaknesses and improved

# Consider other fields of engineering




Mechanical

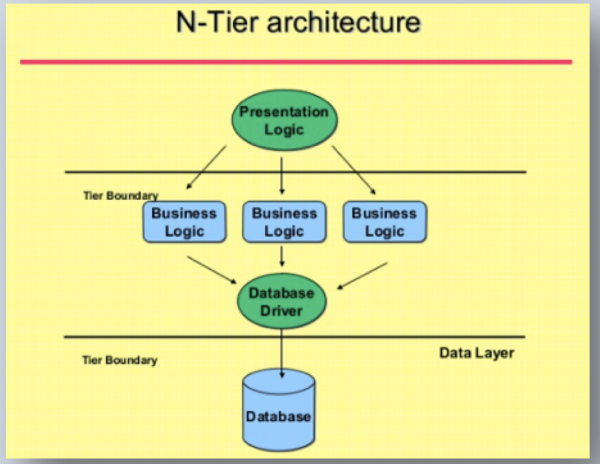


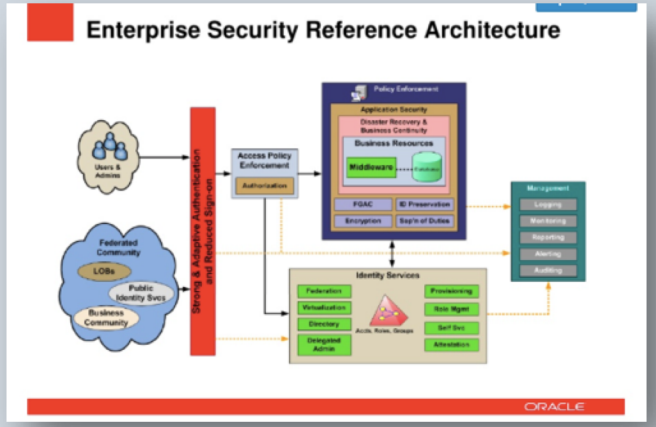

Electrical

Civil

# But what of AppSec?

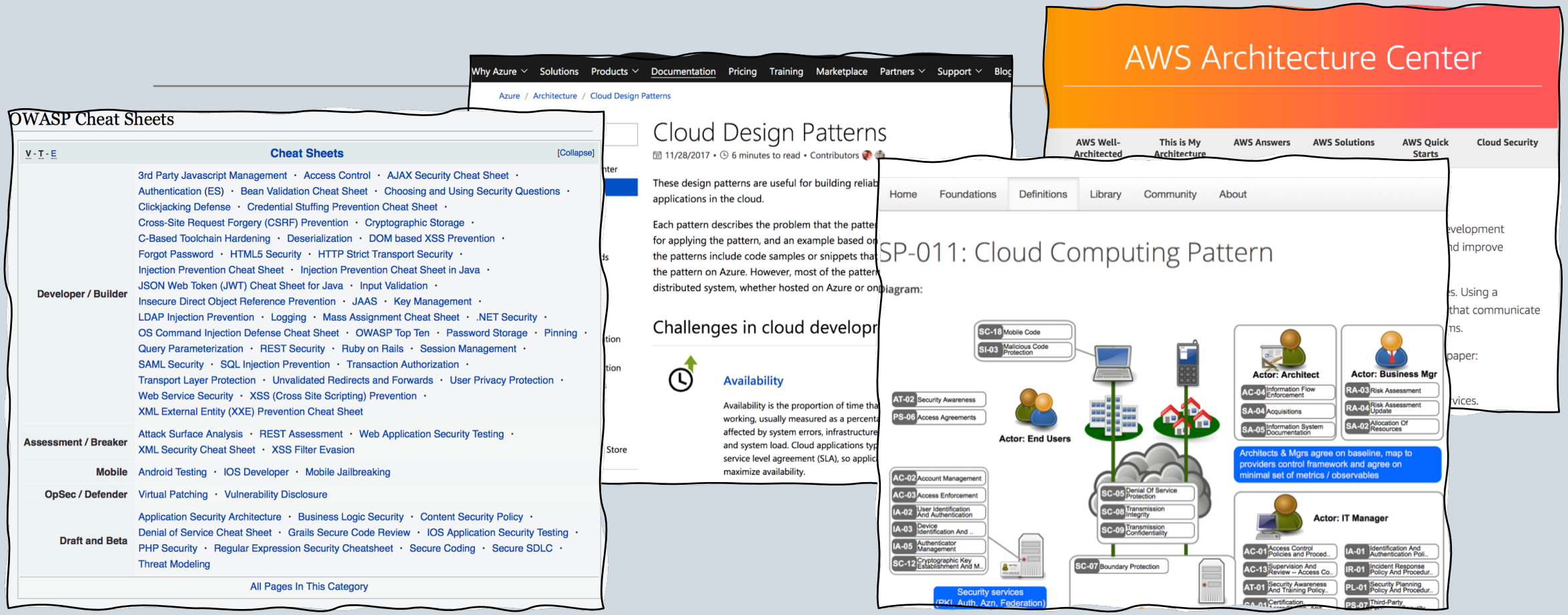
Lots of answers, and we keep making the same mistakes in new contexts




?

# Many good Pattern Catalogs exist…




… but we need one focused on App Sec Design principles.

# Again - Our Iterative Approach:

1. Propose a set of software architecture and software design patterns at various levels of detail
2. Subject those patterns to rigorous analysis, including:
   - Threat Modeling
   - Attack Map / Analysis
   - Live attack trial implementation (RedTeam, Pen Test, etc. – pick your favorite terminology for an intelligent unbounded attacker)
   - Other analysis approaches we may not have thought of here
3. Learn from the results
4. Goto (1)

# Microservices & Patterns

*We'll look at several common patterns involved in:*

The construction of a microservice

The way a set of microservices interact to form an application

How these patterns work together

Pattern-Zero
(point 1)

# A Layered Software Architecture View

*PROPOSED SECURE SOFTWARE DESIGN PATTERNS*

# Problem:  Software Suffers when Confusion exists among Views at Different Altitudes

Solution:  Provide consistency and coordination among these views.

Preferred solution:  Make this consistency repeatable and automate it.

# The Solution: Layered Pattern Stacks as Code (LPSaC)

Use UML diagrams and Microsoft's Layer diagram to flesh out where the security controls go and how they work

Use automated tools (to be discovered/developed) to ensure consistency of the code with each diagram

Now we can bring many groups into sync:

- Architects
- Developers
- Designers
- …

# Layer Diagram:

Diagram properties:

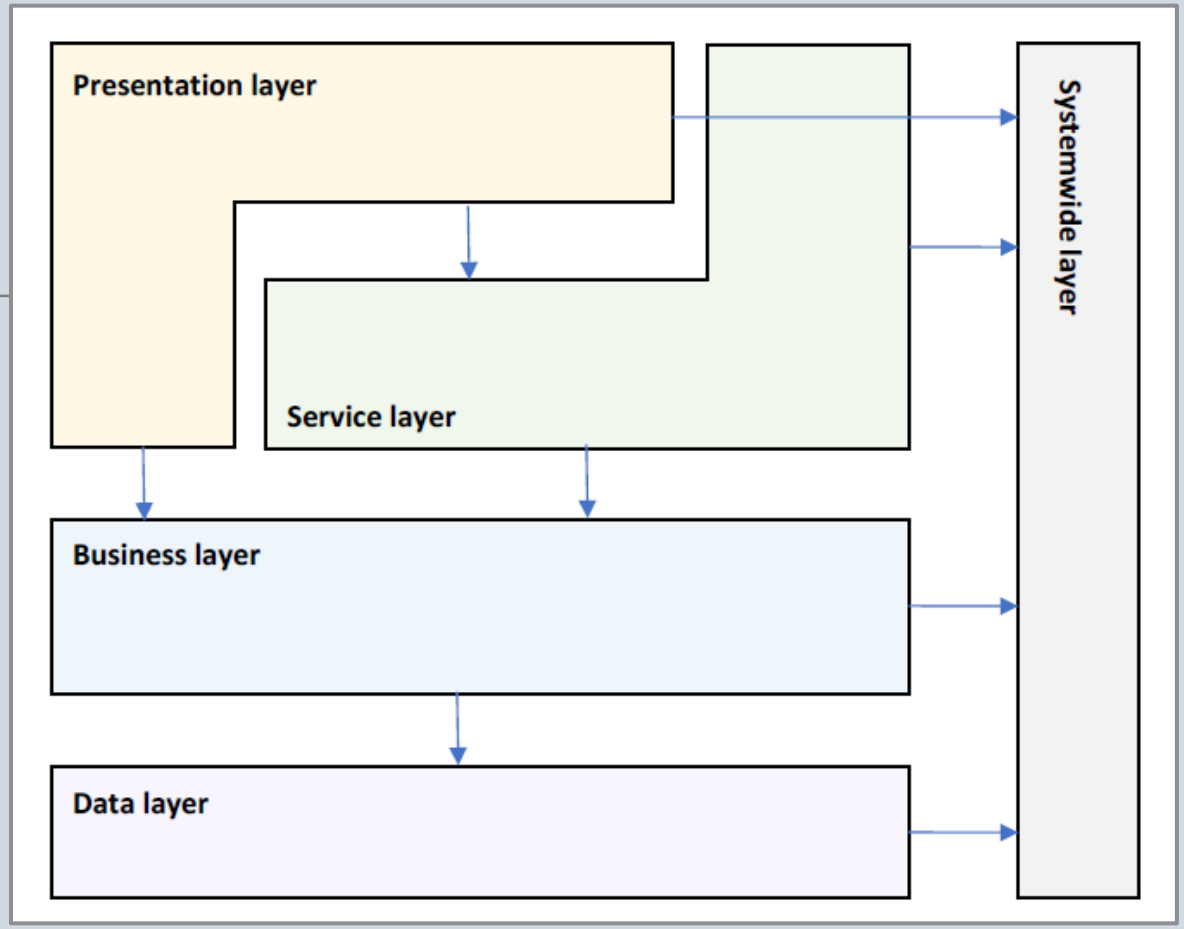
- A high-level system diagram showing areas of concern.
- Proposed by Microsoft, not an official UML diagram

Answers Question:

- What is the overall picture? Where does the software live?

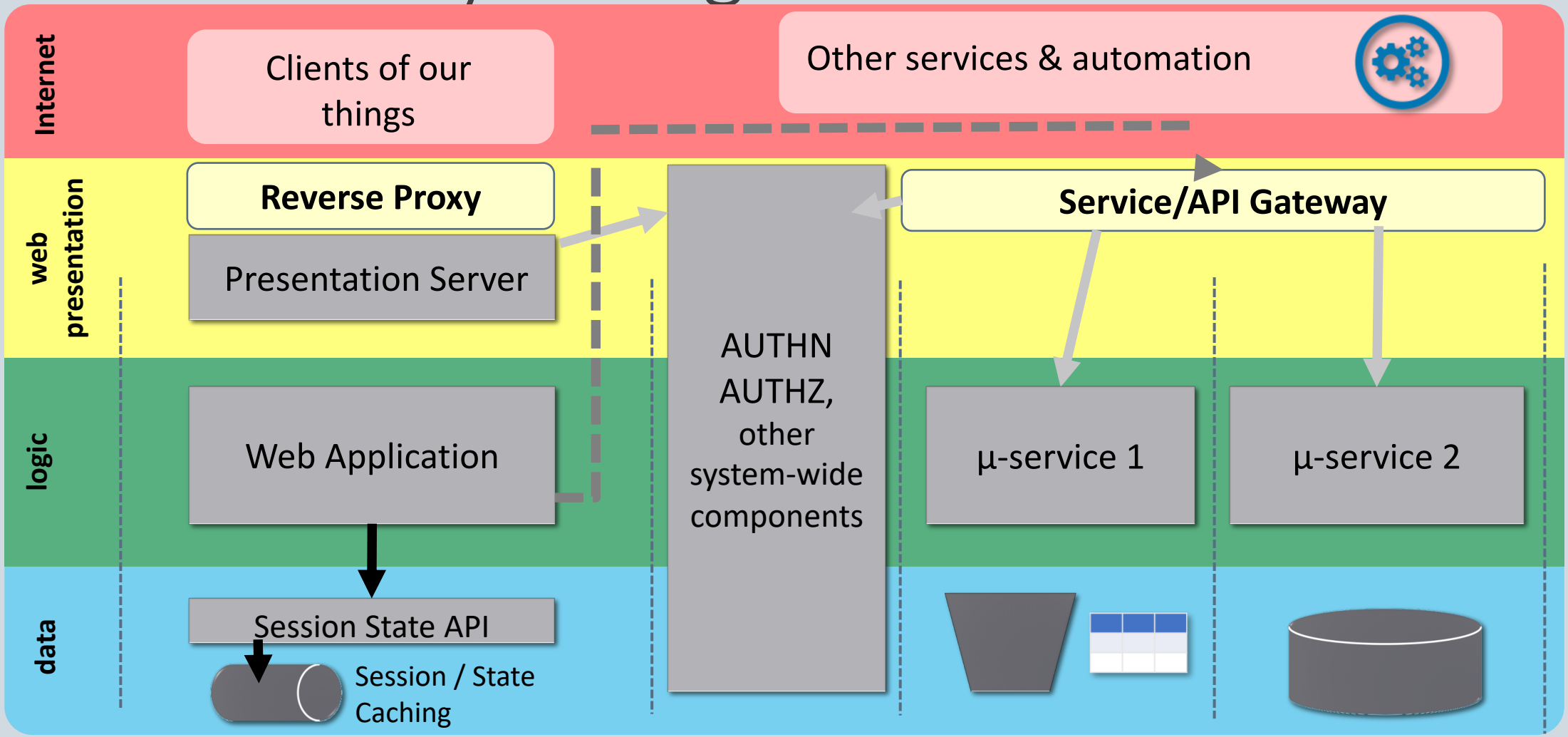
Serves Purpose:

- Depicts an overall map of the relevant Software Entities in order to keep all parties on the same page, and allow ordering of any needed infrastructure.




**See Farragher: https://training.mdfarragher.com/p/learn-how-to-become-an-outstanding-solution-architect**

# Microservice Layer Diagram



# Define “Software Entities” or SW Entities

Examples:

web App

web service

mobile app

API Gateway

Database

# Define Software Component

A (sometimes re-usable) “LEGO Block” of code that can be snapped together with other custom or re-usable blocks of code to create a SW entity.

# High Level App Sec functions of the Service Gateway

*(a.k.a. API Gateway)*

Provides **Security Services** for microservices, such as :

- Single Entry Point pattern→ this is the only way in to the services
- Message-level validation
- Authorization:
  - Is client authorized to talk to this microservice?
  - Is this microservice authorized to talk to the other one?

Keeps the apps simpler, by presenting a **secure façade**, so the app just gets data—it doesn't need to interact directly with or even know about the horde of microservices.

**Routes messages** to microservices, maintains registry of microservices ("Server Side Discovery" with "Service Registry" and "Self Registration.")

**Aggregates responses** to client

Offloads **communication and some configuration** responsibilities to gateway instead of microservice

**Logging, auditing**, health checks of microservices

# UML Sequence Diagram

Diagram properties:

Shows sequence of calls

Shows calling class, called method, and returned type

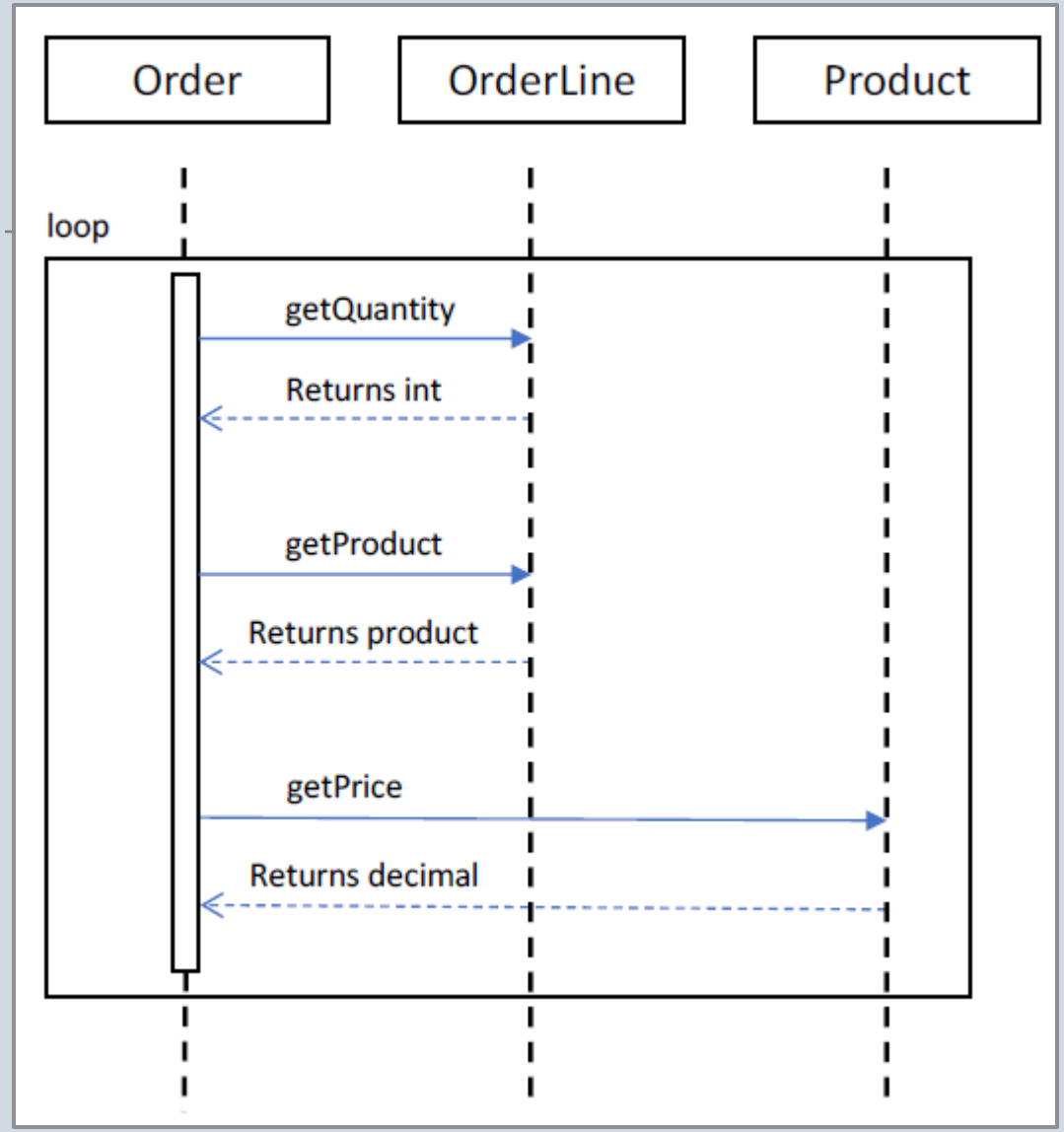
Can depict loops

Answers Question:

How do SW Entities collaborate?

Serves Purpose:

Shows how interactions among different software entities are coordinated to meet requirements for a use case.




See Farragher: https://training.mdfarragher.com/p/learn-how-to-become-an-outstanding-solution-architect

# Activity Diagram

Diagram Properties:

Shows process or workflow

Can show concurrent actions

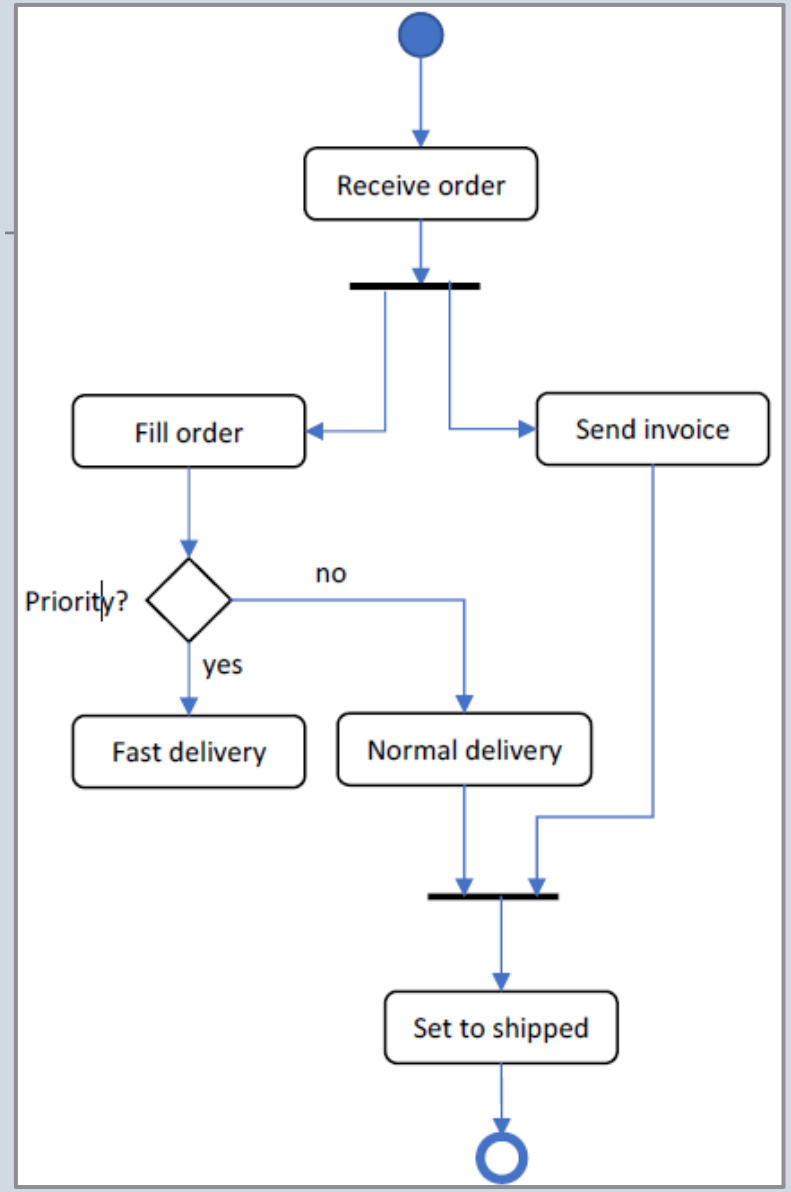
Can be nested

Answers Question:

How do SW Entities collaborate?

Serves Purpose:

Shows how interactions among different software entities are coordinated to meet requirements for a use case.




See Farragher: https://training.mdfarragher.com/p/learn-how-to-become-an-outstanding-solution-architect

# Component Diagram

Diagram Properties:

Shows components

Shows implemented and required interfaces

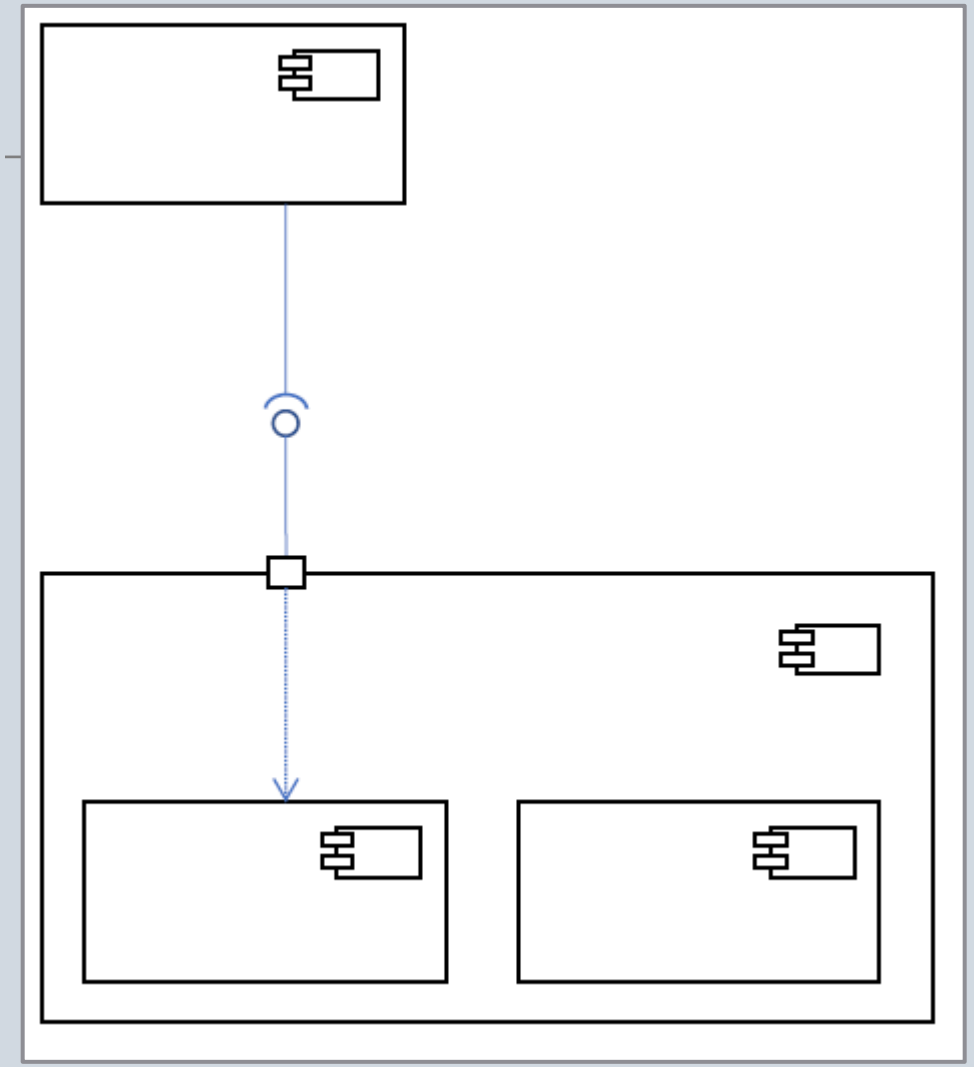
Components can be nested

Answers Question:

Of what custom or reusable LEGO Blocks is this Software Entity composed?

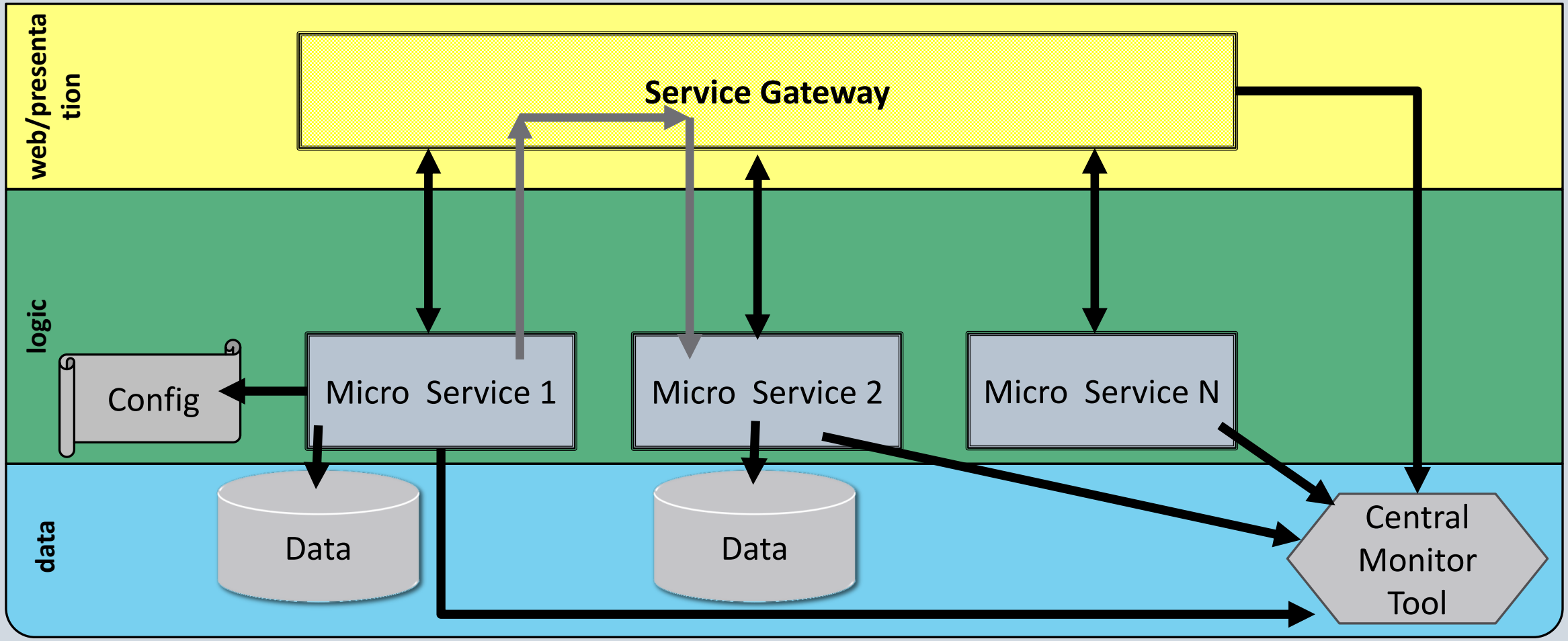
Serves Purpose:

Shows the pieces of software within each entity and the App Sec responsibilities of each.

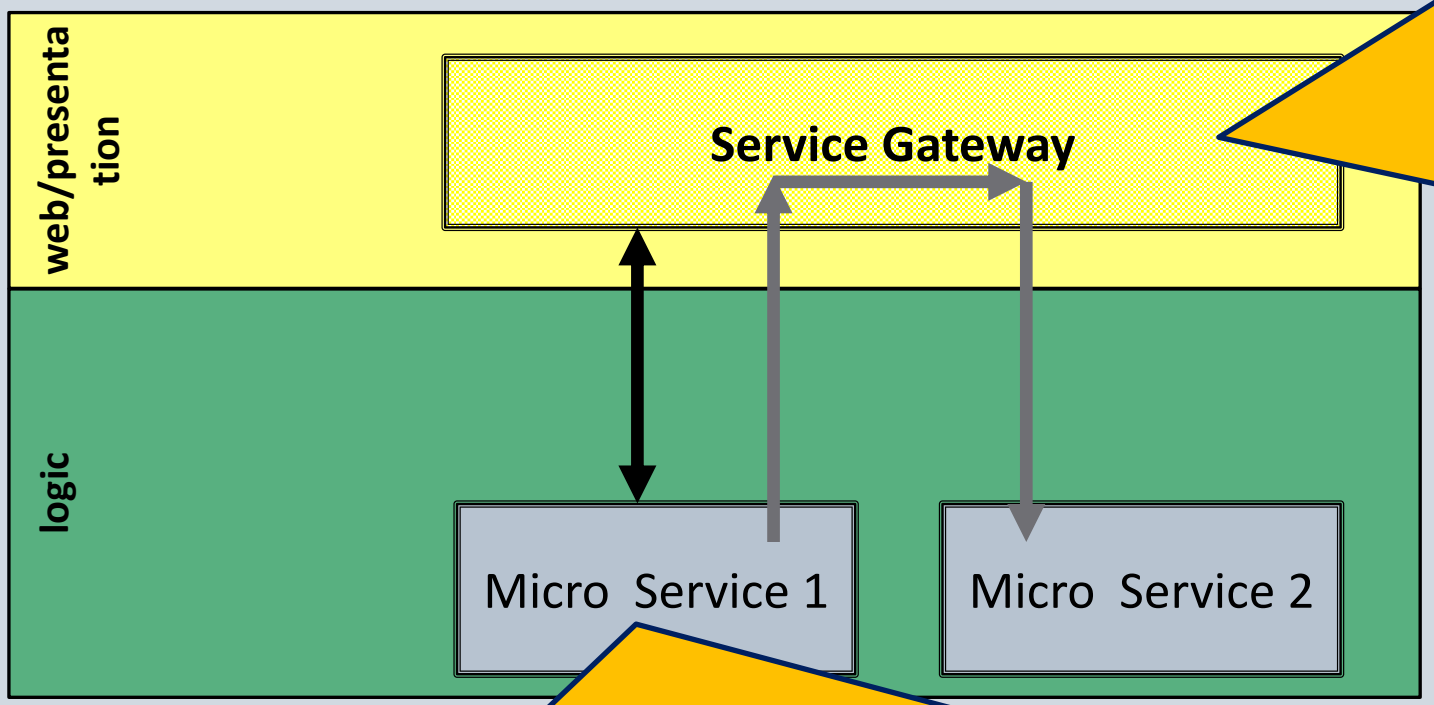


See Farragher: https://training.mdfarragher.com/p/learn-how-to-become-an-outstanding-solution-architect

# Each Micro Service communicates with all these things:



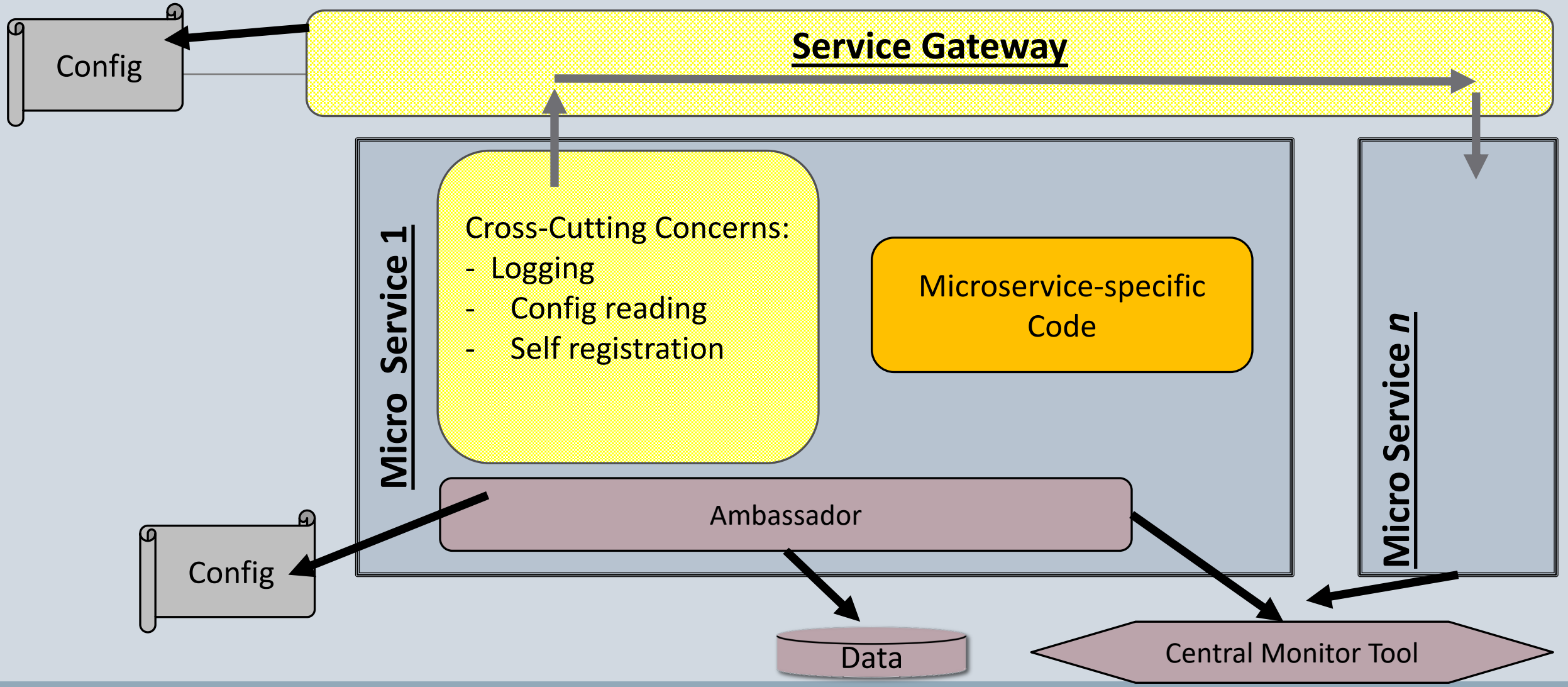
# Look at the App Sec and communications responsibilities of each component to help us map out its sub-components



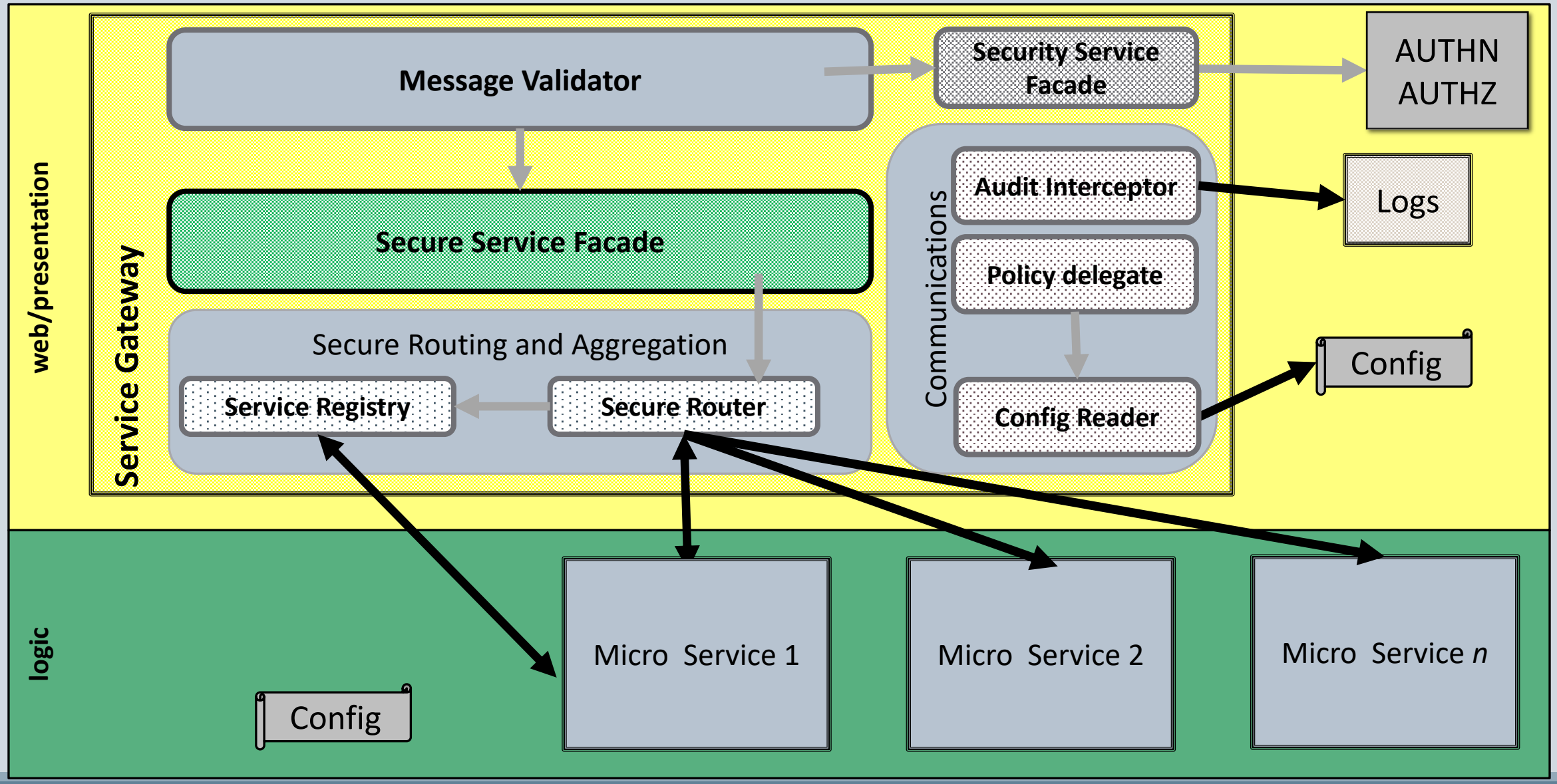

- Single Entry Point
  - Incoming message validation
  - Incoming message Authorization
- Facade for the services below
- Message routing to services
  - Requires router
  - Requires Service Registry
- Response aggregation for client
- Handles functions common to all services, such as:
  - Audit Logging
  - Possibly health checks on services

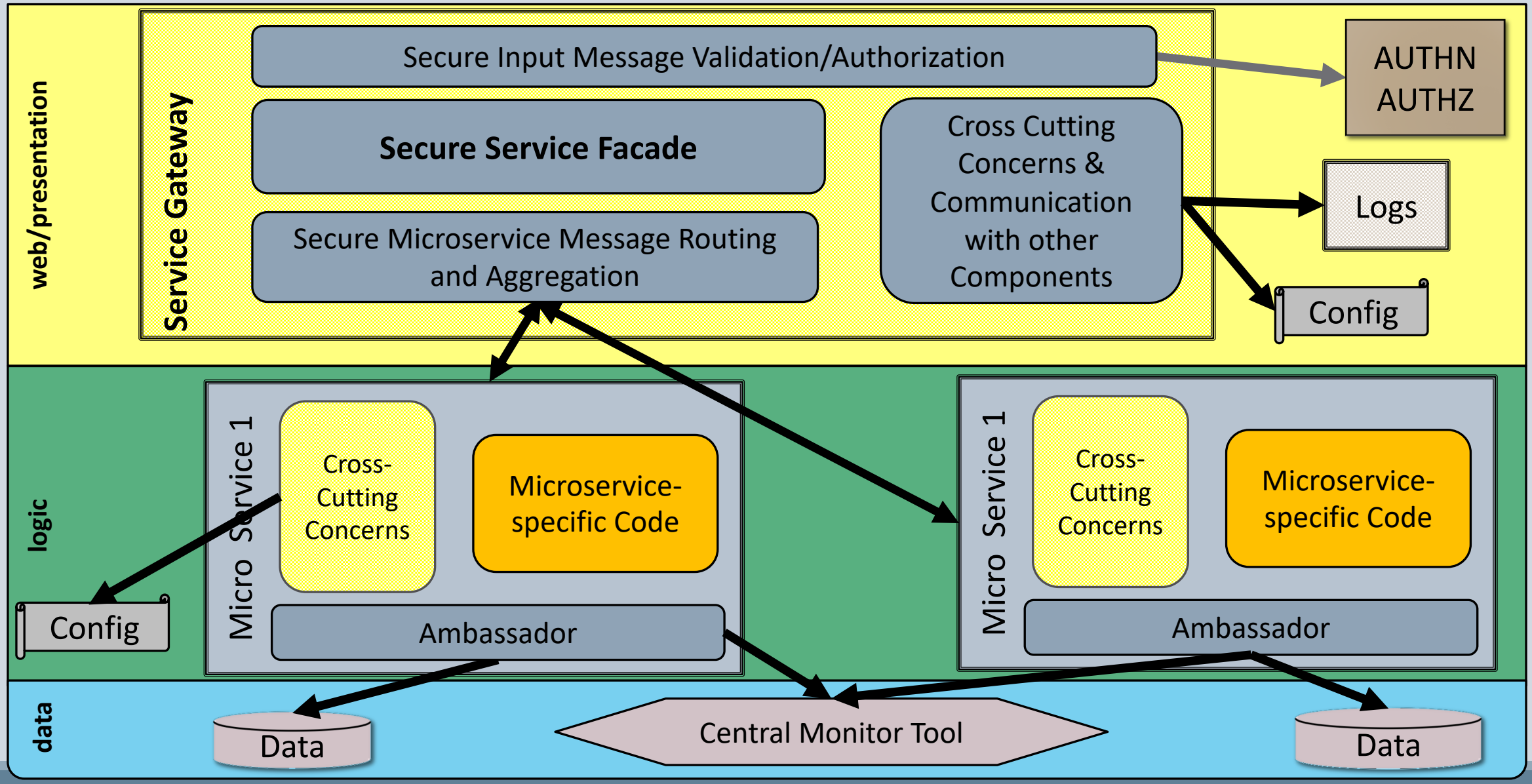
- Simple API for limited set of closely related functions
- Low-level contextual Input validation and Authorization
- Communications to: other services, Service Gateway, config, logging, data store…

# Create components to deliver on each set of responsibilities:

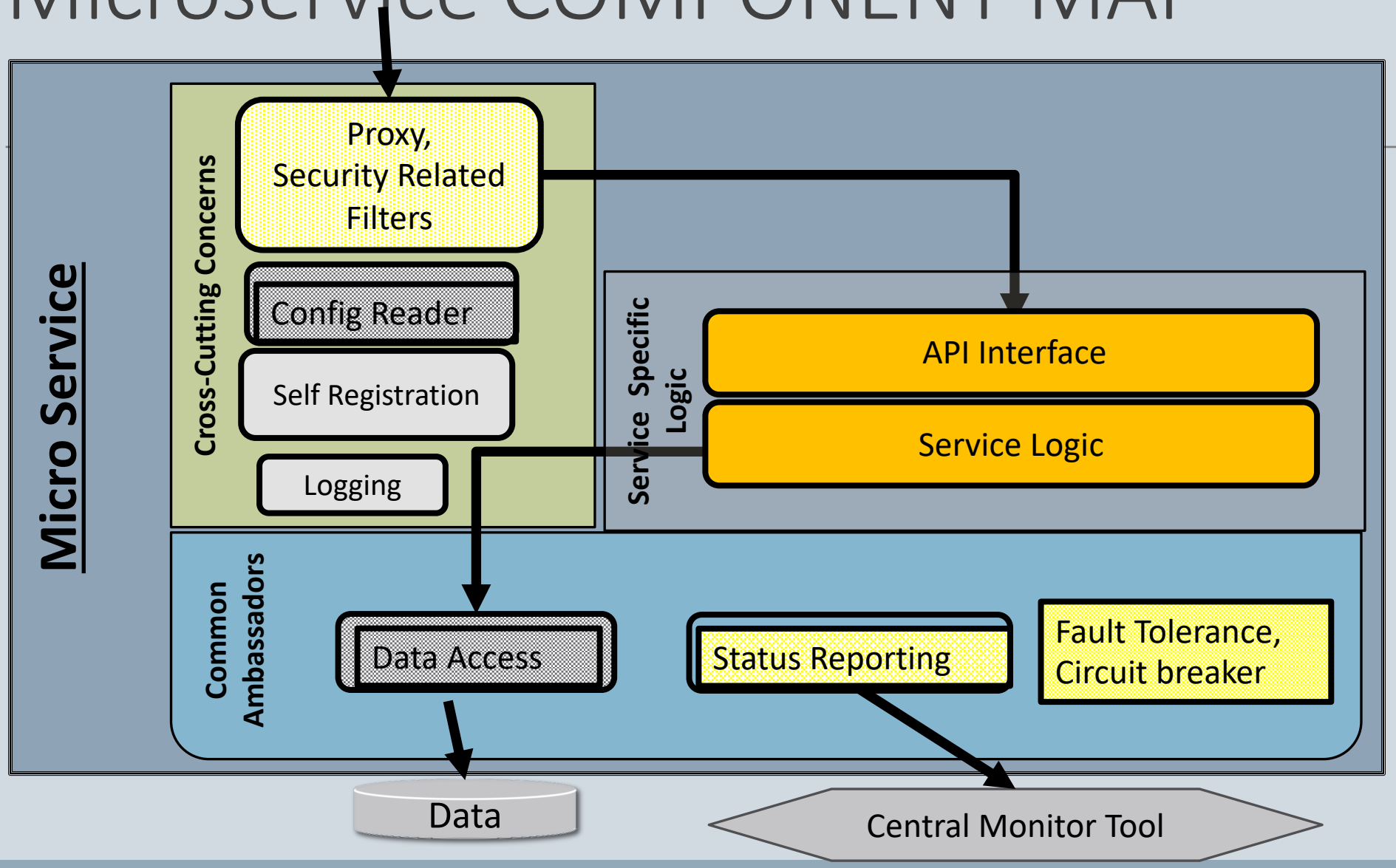


# API Gateway COMPONENT MAP




web/presentation
Service Gateway
Secure Input Message Validation/Authorization
Secure Service Facade
Secure Microservice Message Routing and Aggregation
Cross Cutting Concerns & Communication with other Components
AUTHN AUTHZ
Logs
Config
logic
Micro Service 1
Cross-Cutting Concerns
Microservice-specific Code
Ambassador
Config
Micro Service 1
Cross-Cutting Concerns
Microservice-specific Code
Ambassador
data
Data
Central Monitor Tool
Data

# Microservice COMPONENT MAP



# Class Diagram

Diagram Properties:

Shows classes

Shows methods and fields

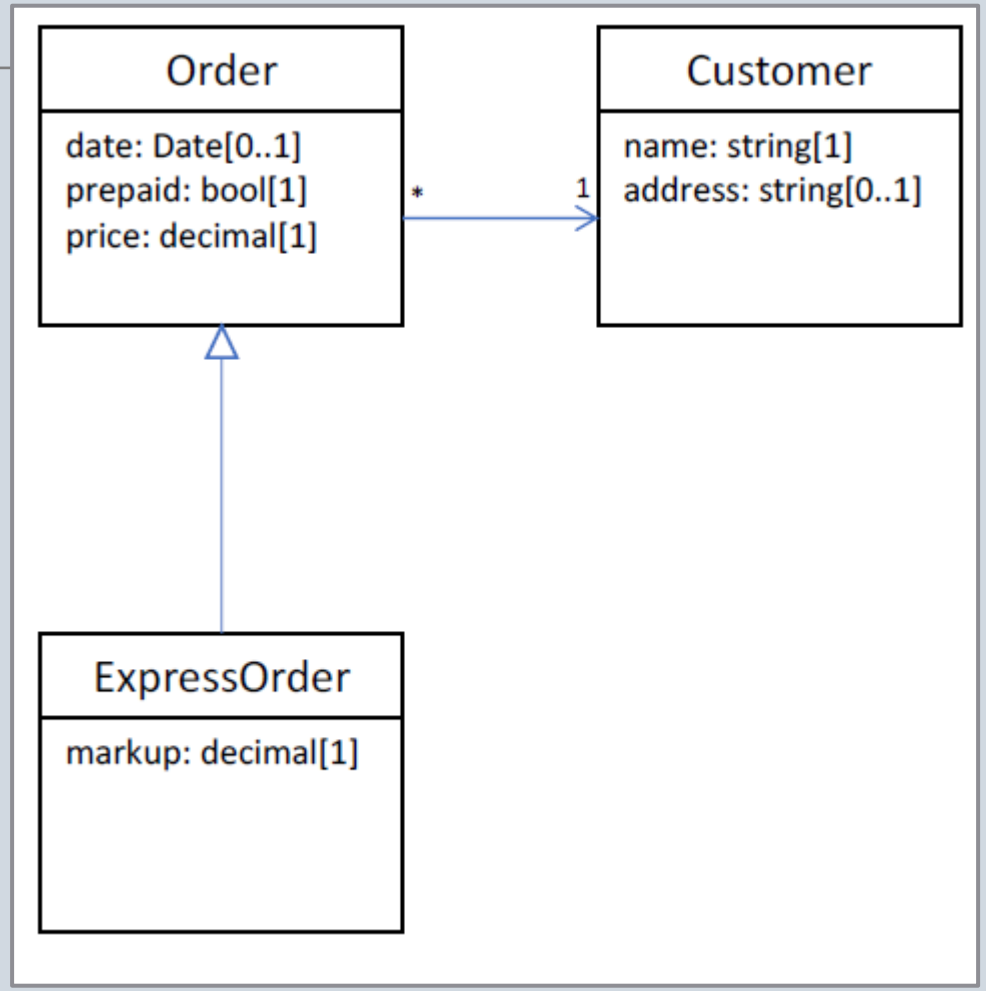
Shows associations, generalizations, and cardinality

Answers Question:

How shall the team organize the code for this lego block (software component)?

Serves Purpose:

Shows what this block will do and how it will do it.




See Farragher: https://training.mdfarragher.com/p/learn-how-to-become-an-outstanding-solution-architect

# Diagrams as Code…

Wouldn’t it be more useful if these diagrams could be used to generate code?

What if the code could later be validated against the patterns to ensure alignment?

How would we do this?

- New language, an idea whose time has come?
- Or, automate handling of these diagrams in an IDE….

# Requirements for Diagrams as Code

Automatic conversion of visual diagram to code

Automatic conversion of code to diagram

**Ability to validate written code against intended diagram**→ testing and Governance

Diagrams under version control

# Visual Studio Ultimate 2017 Provides some of the Needed Support for Diagrams as Code:

It *understands* Layer, Sequential, Activity, Component, and Class Diagrams

It allows conversion of Class diagrams *into code*

It allows conversion of *code into* Class and Layer *diagrams*

It will validate that the code matches the intended Layer diagram

What we need is this support for all the types of diagrams mentioned

- It can perhaps be built out using extensions and T4

# Summary of Diagrams

| Level of Detail | Diagram Name | Answers Question | Purpose of Diagram | Notes |
|---|---|---|---|---|
| On what planet does the software live? | **Use Case Diagram** | What are the functional requirements? | Captures Functional Requirements | Not so actionable without the other diagrams; doesn't show which software entities do the work. |
| What continent? | **Layer Diagram** | What does the software do, and how does it do it? | Overall map to organize all parties Map of Software Entities | Can show divisions of App Sec Responsibilities |
| What Country? | **Sequence Diagram & Activity Diagram** | How do structural elements collaborate? | Show how interactions among various software entities meet App Sec responsibilities | |
| What region? | **Component Diagram** | Of what (reuseable?) blocks is the software composed? | Enumerate App Sec responsibilities for each software component | Components could be nested, requiring addional diagrams |
| What neighborhood? | **Class Diagram** | How to organize the code for a given component? | Shows what each component will do, and how it will be organized | |

# Lessons Learned

Assessing the security responsibilities of software helps determine what components are required

Patters need to be organized hierarchically, to keep all levels in check

Patterns always have consequences: choose patterns with consequences that encourage secure designs

Existing pattern catalogs do not consider security consequences of cataloged patterns, this is a gap and an opportunity for the AppSec community

Infrastructure patterns should not try to solve problems that are fundamentally the responsibility of the software / app

Using infrastructure to solve problems that are the responsibility of the software encourages insecure patterns in the software

# We present this work as Pattern Zero point 1

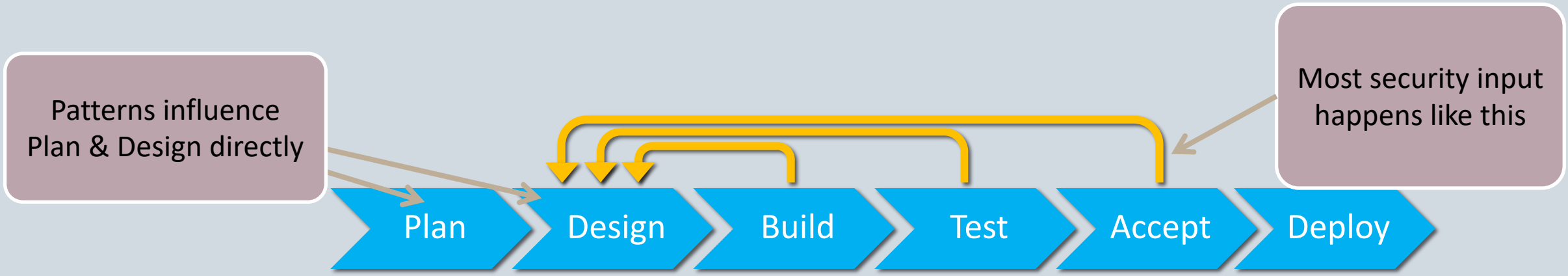
We think we have a workable template for designing software (or at least, something pretty close)

We can map where the security controls should go within software component designs

We can add to the OSA model by

- Indicating which pattern to use for which situation
- Mapping software level security controls to software components

This effort offers the opportunity to push the OWASP controls left, into the design phase.



# Where to go from here

We have proposed a pattern set for the use of microservices in DevSecOps

Time to seek industry collaboration on iterative pattern improvement according to the algorithm we proposed:

1. ✓ Propose a set of software architecture and software design patterns at various levels of detail
2. → Subject those patterns to rigorous analysis, including:
   - Threat Modeling
   - Attack Map / Analysis
   - Live attack trial implementation (RedTeam, Pen Test, etc. – pick your favorite terminology for an intelligent unbounded attacker)
   - Other analysis approaches we may not have thought of here
3. Learn from the results
4. Goto (1)

# Thanks!  Q& A time!

# References


Mark Farragher “How to become an Outstanding Solution Architect”

Micro service architecture: http://microservices.io/patterns/microservices.html

Micro service API Gateways: http://microservices.io/patterns/apigateway.html

Azure Microservices architecture: https://azure.microsoft.com/en-us/blog/design-patterns-for-microservices/

Open Security Architecture: http://www.opensecurityarchitecture.org/cms/library

Other talks on Patterns: https://www.owasp.org/images/1/11/Vanhilst_owasp_140319.pdf

https://msdn.microsoft.com/en-us/library/ee658117.aspx

http://pubs.opengroup.org/architecture/togaf8-doc/arch/chap28.html

http://www.oreilly.com/programming/free/files/software-architecture-patterns.pdf