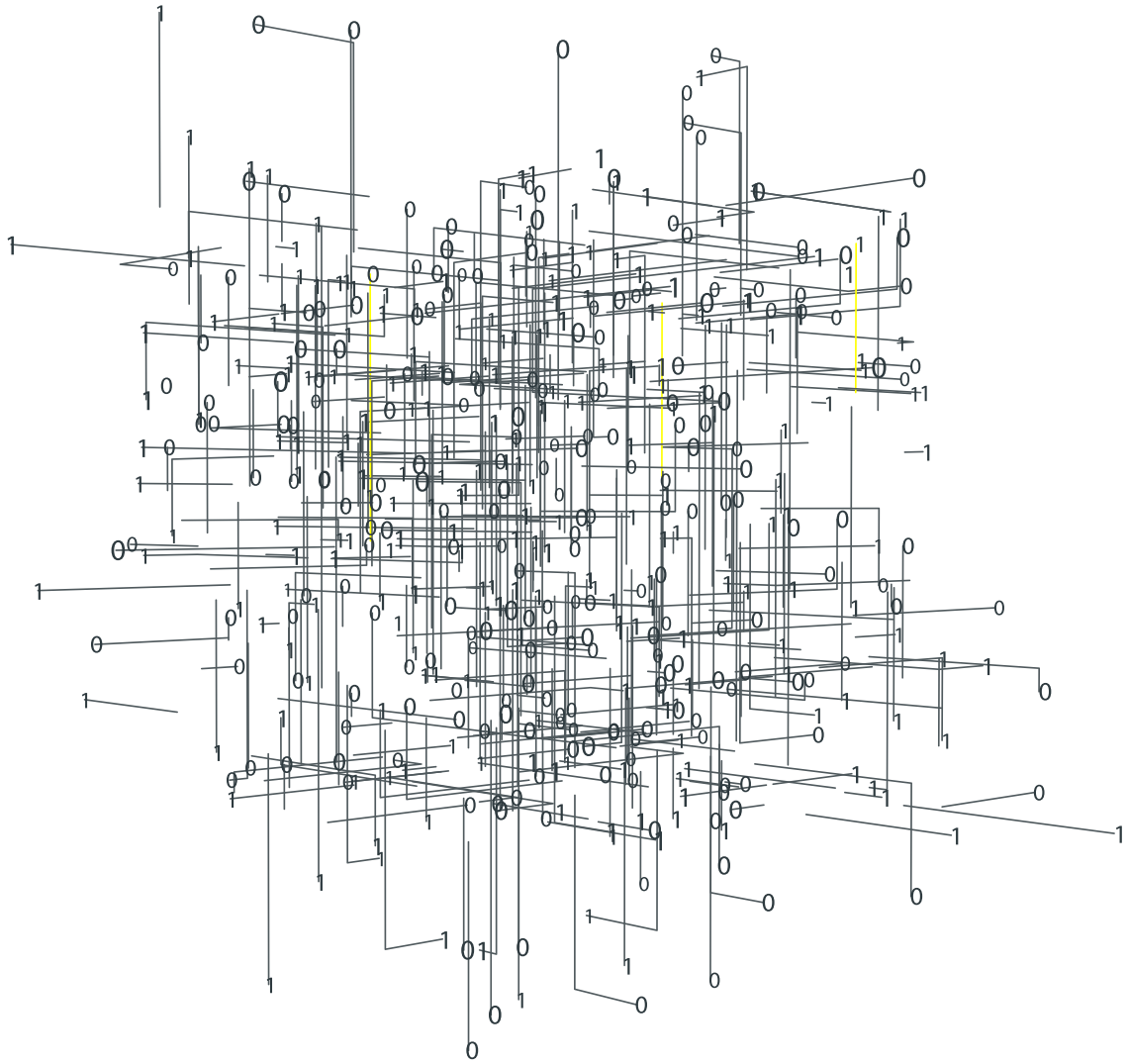
# 2DT902

## Avancerade koncept

API:er o Driftssättning



Linnæus University

# Dagens föreläsning

API:er

”composability”

Business agility

Open API

CD/CI

omgivningar

”deployability”

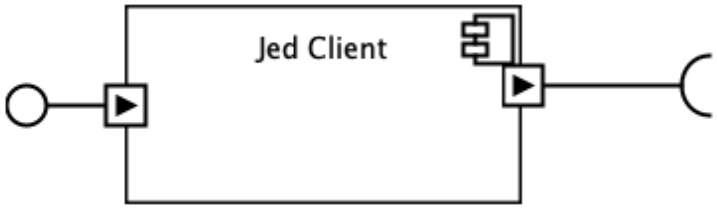
Mönster & taktiker

“Composability,
Pieces fit, systems align,
Deploy with a breeze.”

— ChatGPT 4o

# API:er i centrum

API—“application programming interface”




”Design by contract”

Orkestrering av system

# Vad är ett kontrakt??

```
preconditions
postconditions
invariants
```

Det finns två parter:

- Kund som begär en tjänst
- Leverantör som tillhandahåller tjänsten

Kontraktet är avtalet mellan kunden och leverantören

Två viktiga egenskaper hos ett kontrakt

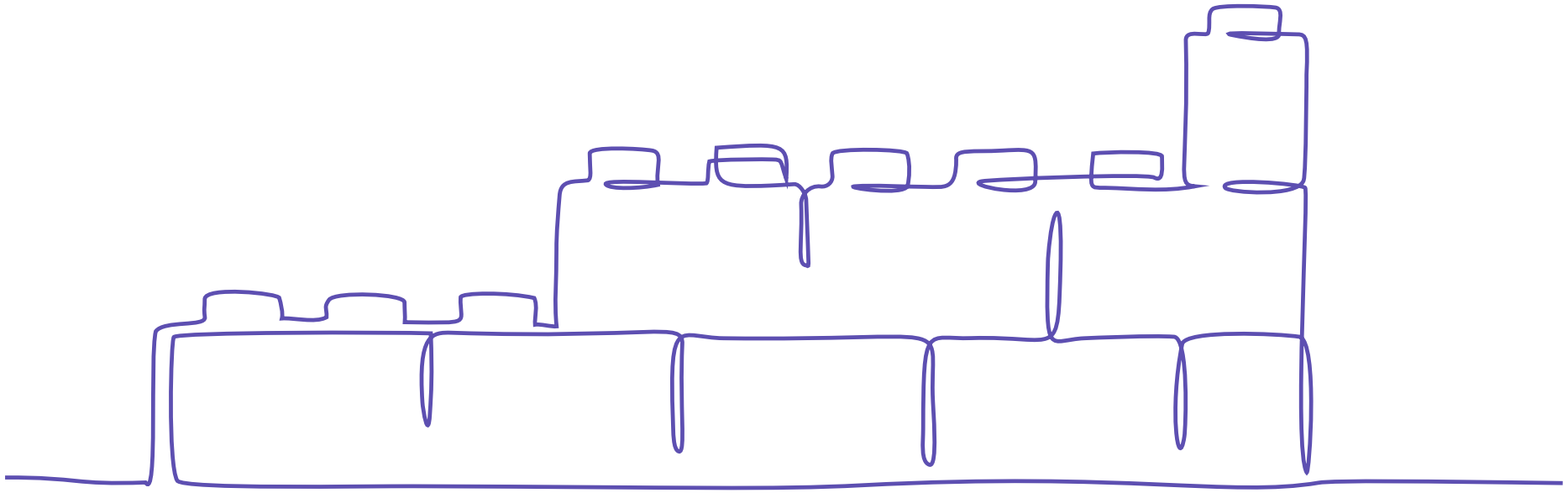
- Varje part har förväntningar på avtalet och är beredd att ta på sig vissa skyldigheter för att uppfylla dessa.
- Fördelar och skyldigheter dokumenteras i ett kontraktsdokument

Fördelen för kunden är leverantörens skyldighet, och vice versa.

# Komposabilitet

## Composability

"is a system design principle that deals with the inter-relationships of components. A highly composable system provides components that can be selected and assembled in various combinations to satisfy specific user requirements".



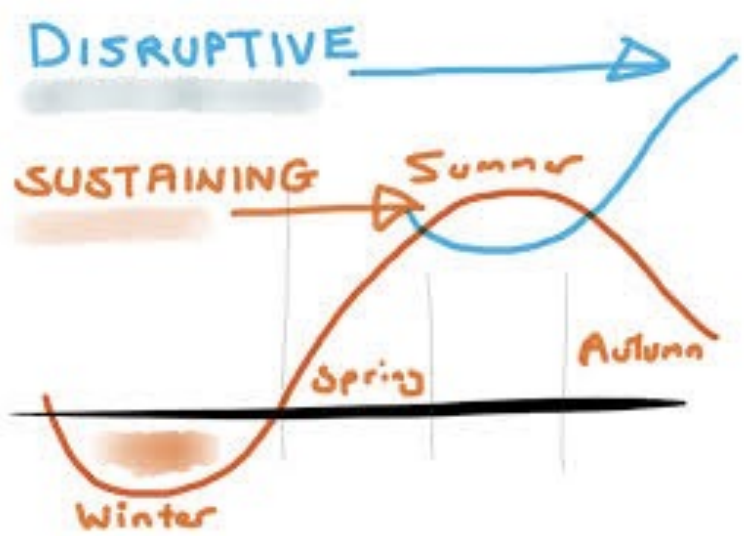


Source: wikipedia

# Affärsagilitet

## Från projekt till produkter

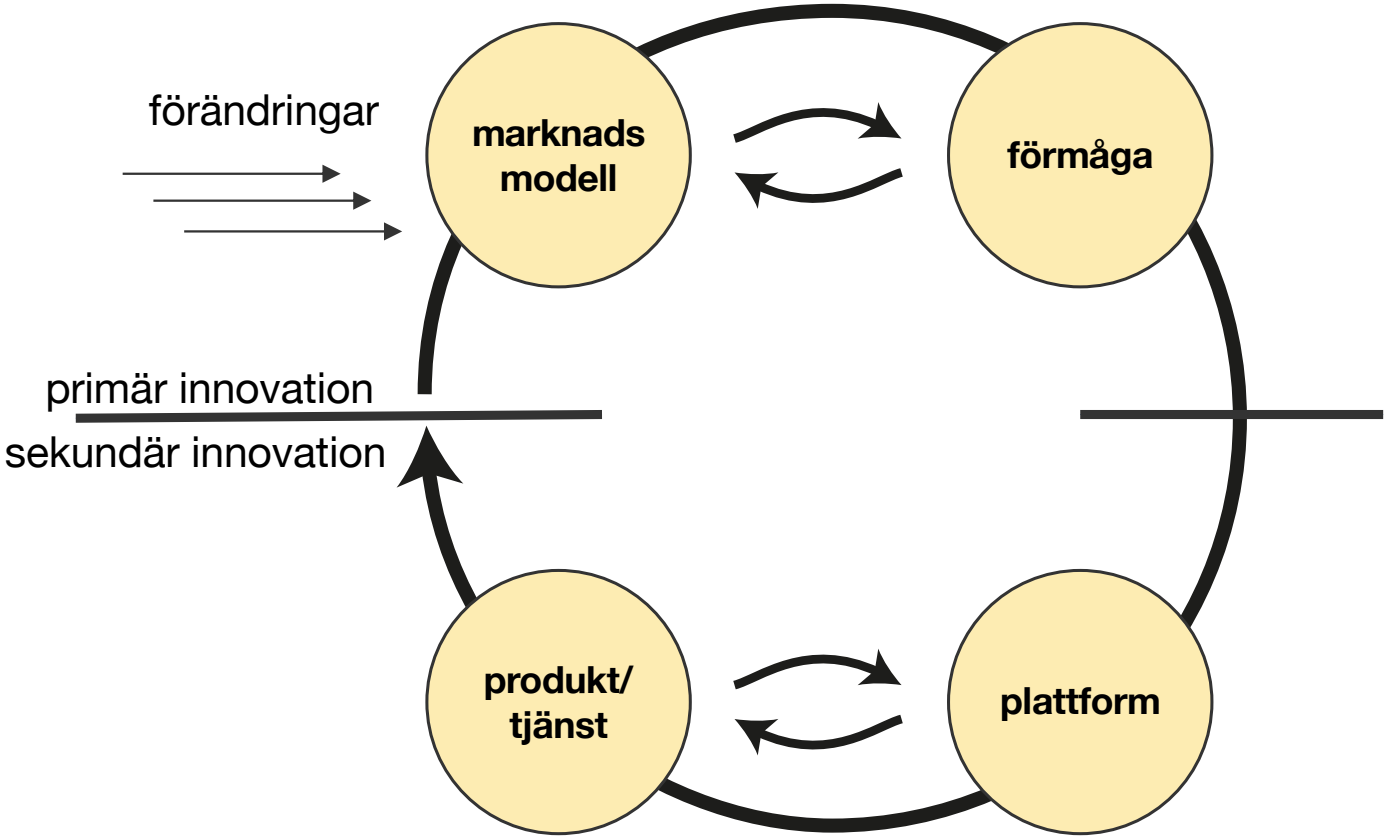
”Business agility is the ability of an organization to sense changes internally or externally and respond accordingly in order to deliver value to its customers.”




Agila manifestet

Reagera på förändringar istället för att följa en plan

# Affärsagilitet



# Affärsagilitet - Evalitetsattribut

**”Enables or inhibits quality”**

Komposabilitet

Driftssättbarhet - en egenskap hos ett system som anger att den kan driftsättas – det vill säga allokeras till en driftsmiljö och köras – inom en förutsägbar tid och med en acceptabel mängd arbete

Testbarhet - hur enkelt, effektivt och ändamålsenligt en applikation kan testas

Tillgänglighet - Sannolikheten för att ett system inte är trasigt eller repareras när det behöver användas.

Skalbarhet - förmågan hos ett system för att hantera en växande mängd arbete.

# API-centrerad design

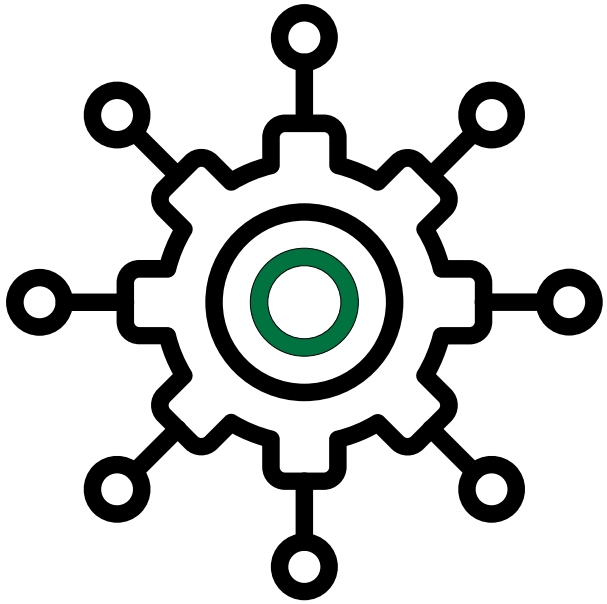
Från ett API

för kommunikation

och integration

av ett system

som är distribuerat



Till flera API:er

för återanvändning

som tjänster

i flera system

ofta över webben

**API:er är nyckelelement i moderna mjukvaruarkitekturer**

# API:er och komposabilitet

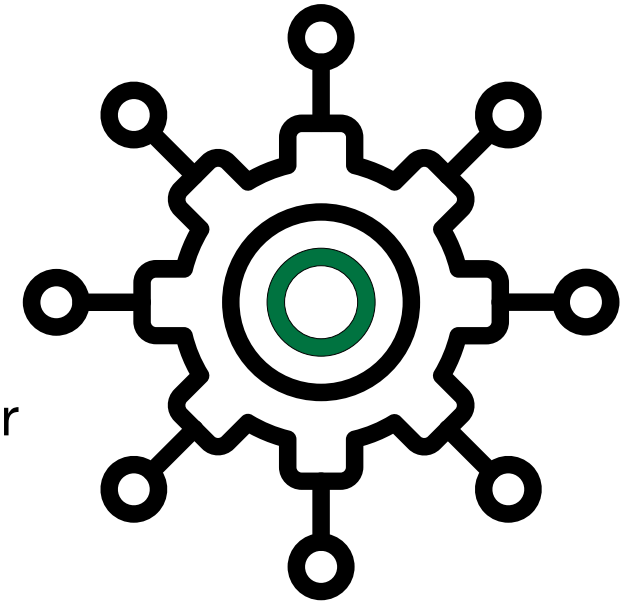
**API:er möjliggör komposabilitet**

Publika API:er

Erbjuder tjänster

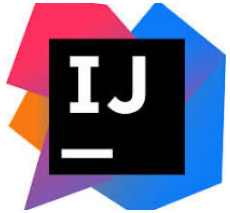
Eller utökningar till plattformar

Business APIs



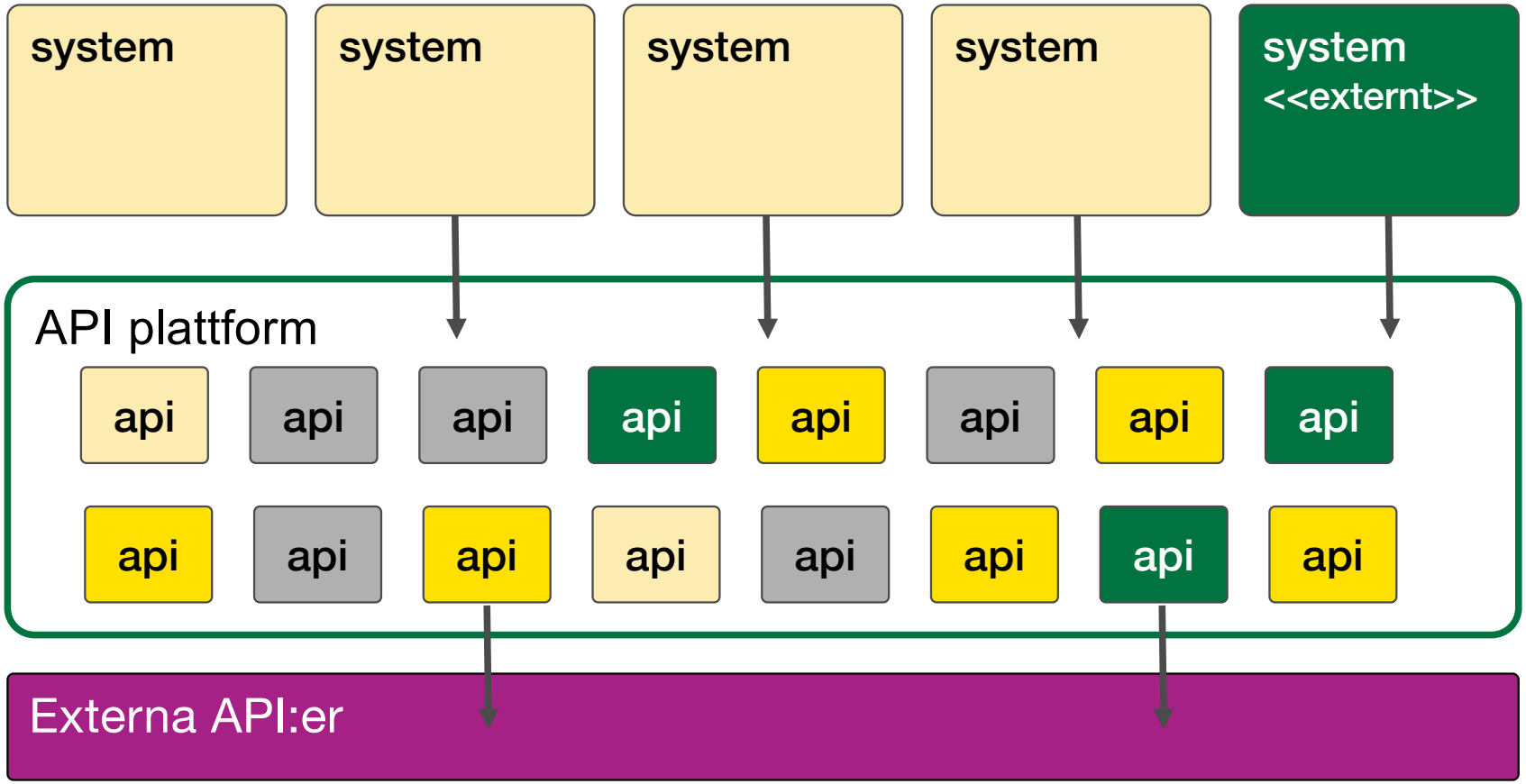
**Exempel**

Tjänst




Plattform

# API baserade plattformar



# Olika API typer

Remote Procedure Call, RPC – Klienter anropar funktioner på en servern.

Remote Method Invocation, RMI – RPC med objekt!

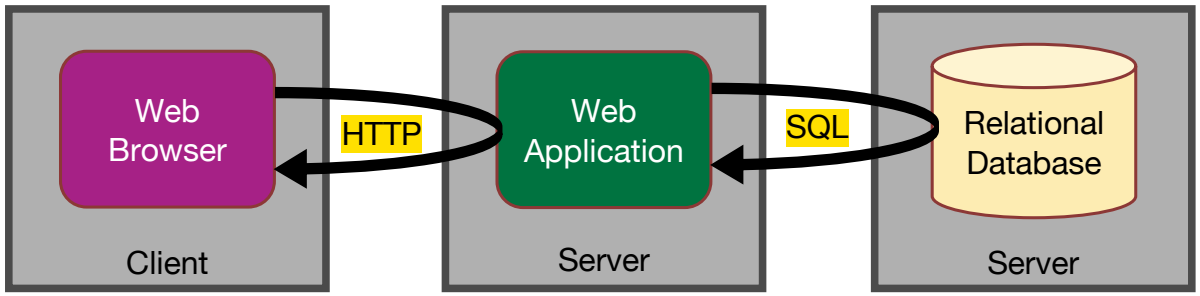
| | | |
|---|---|---|
| **SOAP API** | Ett protokoll som använder XML för att formatera meddelanden | PayPal SOAP API, Salesforce API |
| **REST API** | Bygger på HTTP metoder, tillståndslöst, returnerar typisk JSON or XML. | GitHub API, Spotify API |
| **GraphQL API** | Klienter ställer ”frågor” på grafstrukturer. | GitHub GraphQL API, Shopify API |
| **WebSocket API** | Möjliggör full-duplex kommunikation i realtid. | Binance WebSocket API, Slack API |

# Vad är REST?

En *arkitekturstil* för distribuerade “hyper-mediasystem” som Roy Fielding beskrivs första gången år 2000.

Stil: Mönster och begränsingar:

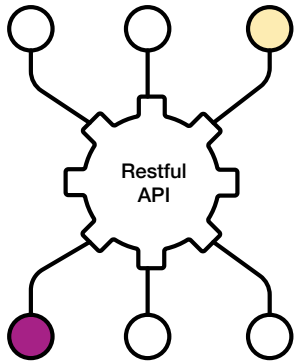
1. Client - Server
2. Stateless
3. Cache
4. Uniform Interface
5. Layered System
6. Code-On-Demand



# REST?

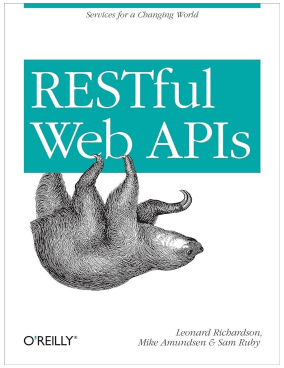
*The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through the application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.*

R. Fielding



# Restfulness

## Richardson's Maturity Model




| Level | Key Concept | Description |
|---|---|---|
| **Level 0** | The Swamp of POX | Uses HTTP as transport only. No resources, only one endpoint for all actions. |
| **Level 1** | Resources | Resources (nouns) are introduced, but HTTP methods are still misused. |
| **Level 2** | HTTP Verbs (Methods) | Proper use of HTTP methods (GET, POST, PUT, DELETE) and status codes. |
| **Level 3** | Hypermedia Controls (HATEOAS) | Hypermedia links guide clients on what to do next, allowing dynamic exploration. |

# API:er som använder HTTP

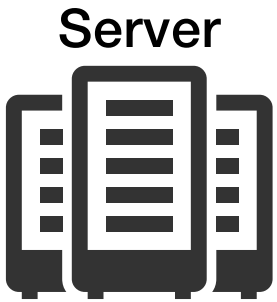
Använder URIs för att identifiera resurser.

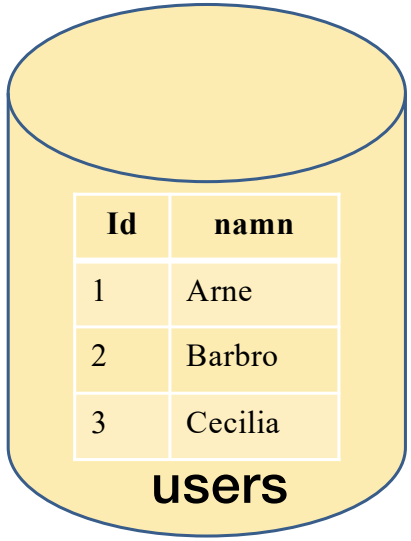
Använder HTTP Metoder för att specificera åtgärd:

- Create: POST
- Retrieve: GET
- Update: PUT
- Delete: DELETE

Använder HTTP statuskod för att indikera utfall (t.ex fel).

# Vad innebär REST?


**GET** /users/2
...

**Tillståndsändring.**

```
{"id": 2, ”namn": ”Babsan"}
```

```
{"id": 2, "namn": ”Barbro"}
```

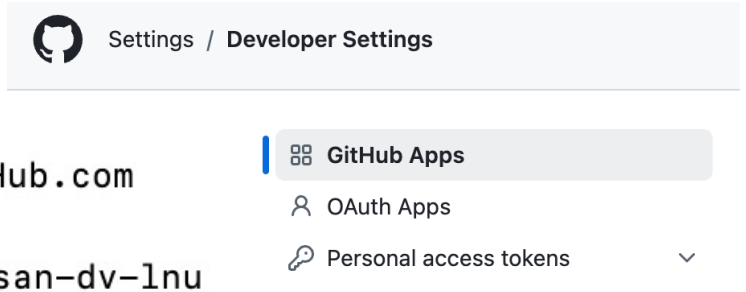
**PUT** /users/2
```
{"id": 2, ”namn": ”Babsan"}
```

# REST API – Exempel

Github

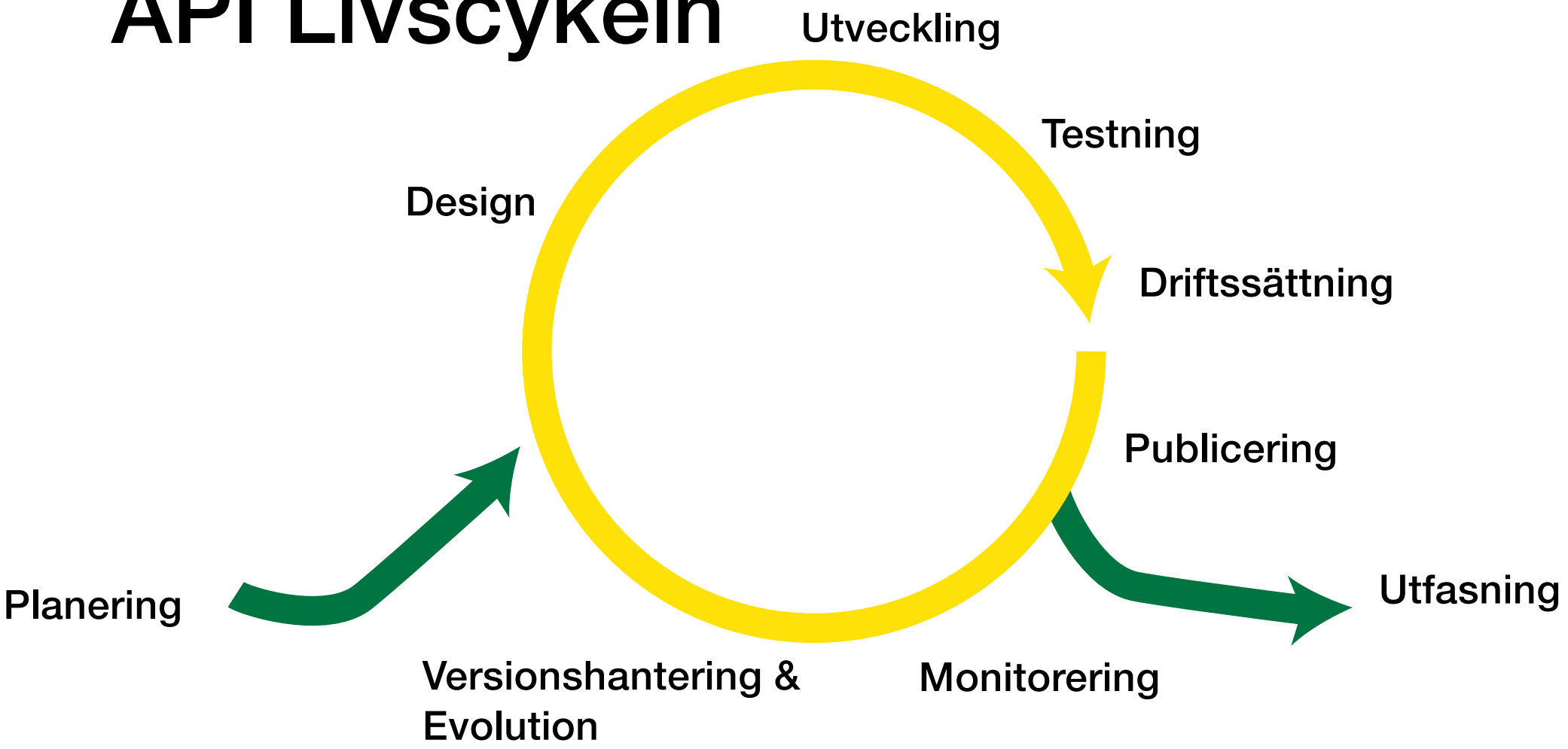
```
janmsi@MB-01009 ~ % curl https://api.GitHub.com

~ % curl https://api.github.com/users/jesan-dv-lnu
```




**Steg 1:** Logga in på ditt GitHub konto.
**Steg 2:** Sök upp **Personal access tokens**.
**Steg 3:** Skapa ett nytt token.
**Steg 4:** Bekräfta med ditt lösenord.
**Steg 5:** Lägg till en beskrivning
**Steg 6:** Välj samtliga ”scopes”
**Steg 7:** Slutligen tryck på generera nytt token.

# API Livscykeln



# Design av en API plattform.

**Förutom att säkerställa att ett API levererar den avsedda funktionaliteten behöver andra aspekter beaktas vid utformningen.**

Syfte och tillräcklig granularitet

Användarperspektiv och återanvändning. Till exempel måste API:er som skall användas i olika system utformas för återanvändning.

Följ etablerade designprinciper för aktuell API typ

Till exempel, endast API:er som är på nivå 3 kan anses vara RESTful.

Stöd API evolution

”*Breaking & non-breaking changes*”. API-versionshantering är nödvändig för att hantera (kommande) ändringar i kontraktet.

Felhantering

Säkerhet

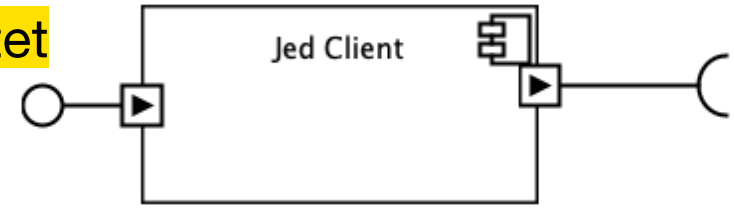
# Specificera ett API

**API-specifikation består av ett antal arkitektursignifikanta beslut**

”Design by contract”

Återanvändning

Funktionalitet




Säkerhet

Granularitet

Felhantering

API evolution

# OpenAPI

**En standard för http baserade API:er**

Underlättar för utvecklare att beskriva och dokumentera API:er
på ett strukturerat och enhetligt sätt.

Underlättar utvecklingsprocessen då verktyg kan automatisera och
skapa en tydlig dokumentation för API:er.

Strukturen ger ett ”standardiserat sätt” vilket ökar sannolikheten att API
är lätt att förstå, använda och underhålla, både internt och externt.

# Open API Exempel

**Delar i specifikationen:**

**Info –** Metadata om API - som namn, beskrivning och version.

**Paths –** Definierar de olika endpoints(vägar) som API:et erbjuder, inklusive HTTP-metoder (GET, POST, etc.) och svarskoder.

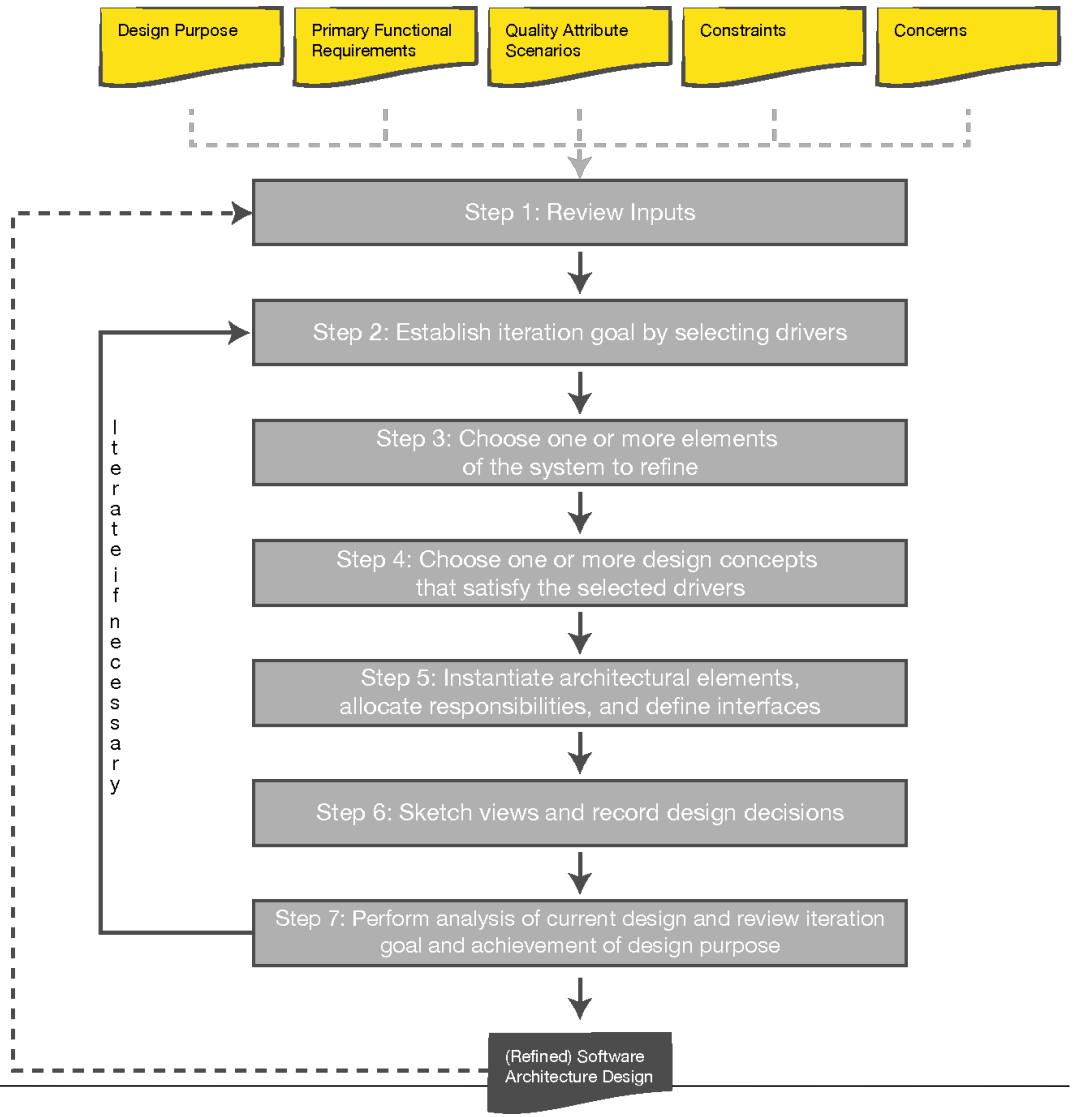
**Responses –** Beskriver svar som API kan returnera, inklusive statuskoder och datatyper.

**Request Body –** Definierar vilken data som kan skickas till API:et, exempelvis för POST- eller PUT-metoder.

# Open API Exempel

```yaml
openapi: 3.0.0
info:
  title: Simple API
  description: Ett enkelt exempel på en OpenAPI-specifikation
  version: 1.0.0
paths:
  /users:
    get:
      summary: Hämta alla användare
      responses:
        '200':
          description: En lista över användare
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
    post:
      summary: Skapa en ny användare
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                name:
                  type: string
                  example: John Doe
      responses:
        '201':
          description: Användare skapad
```

# API:er & ADD


Design Purpose
Primary Functional Requirements
Quality Attribute Scenarios
Constraints
Concerns
Step 1: Review Inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Perform analysis of current design and review iteration goal and achievement of design purpose
Iterate if necessary
(Refined) Software Architecture Design

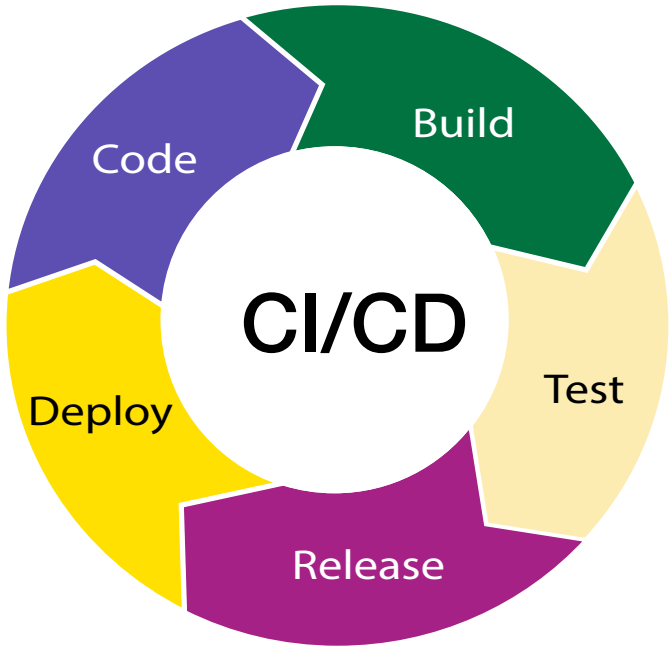
# Driftssättning

Linnæus University

# Driftssättning

**Deployability** refers to a property of software indicating that it may be deployed—allocated to an environment for execution—with predictable time and effort.

Om driftssättning sker automatiskt talar man om continous deployment

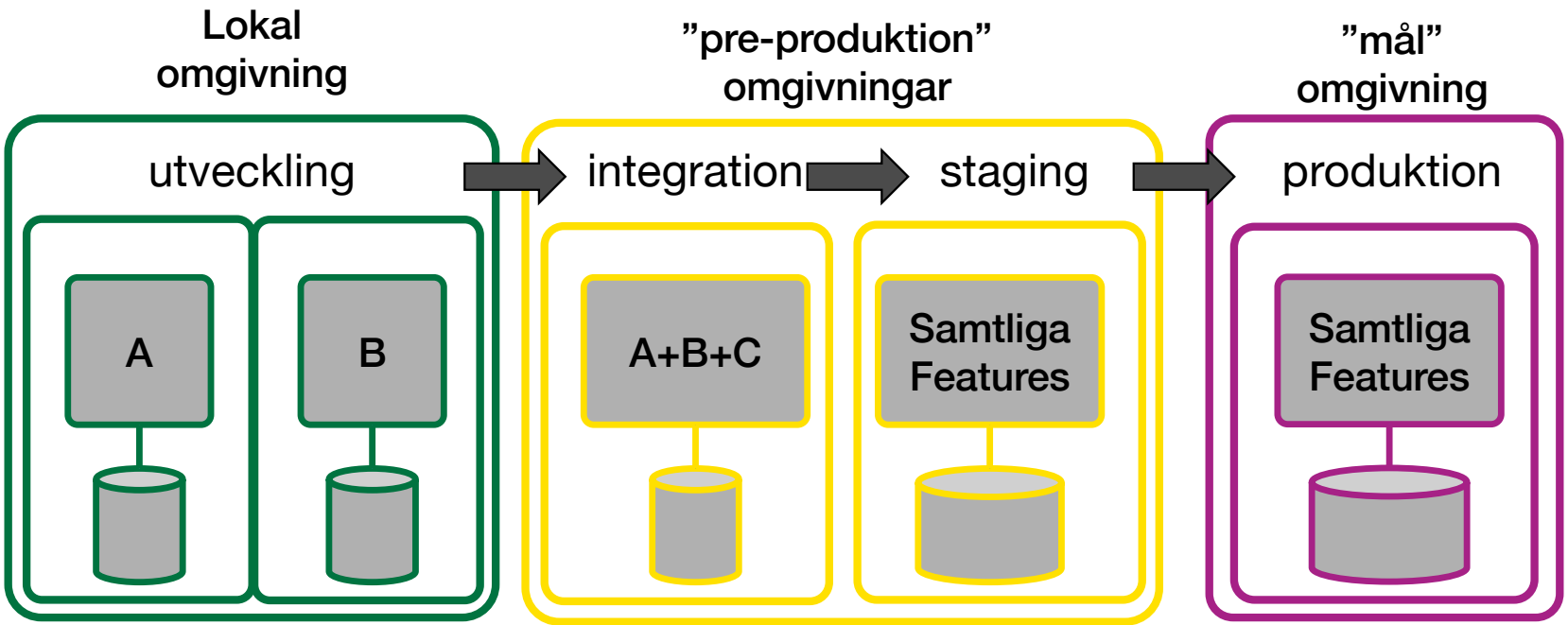


# Arkitekturfrågan!

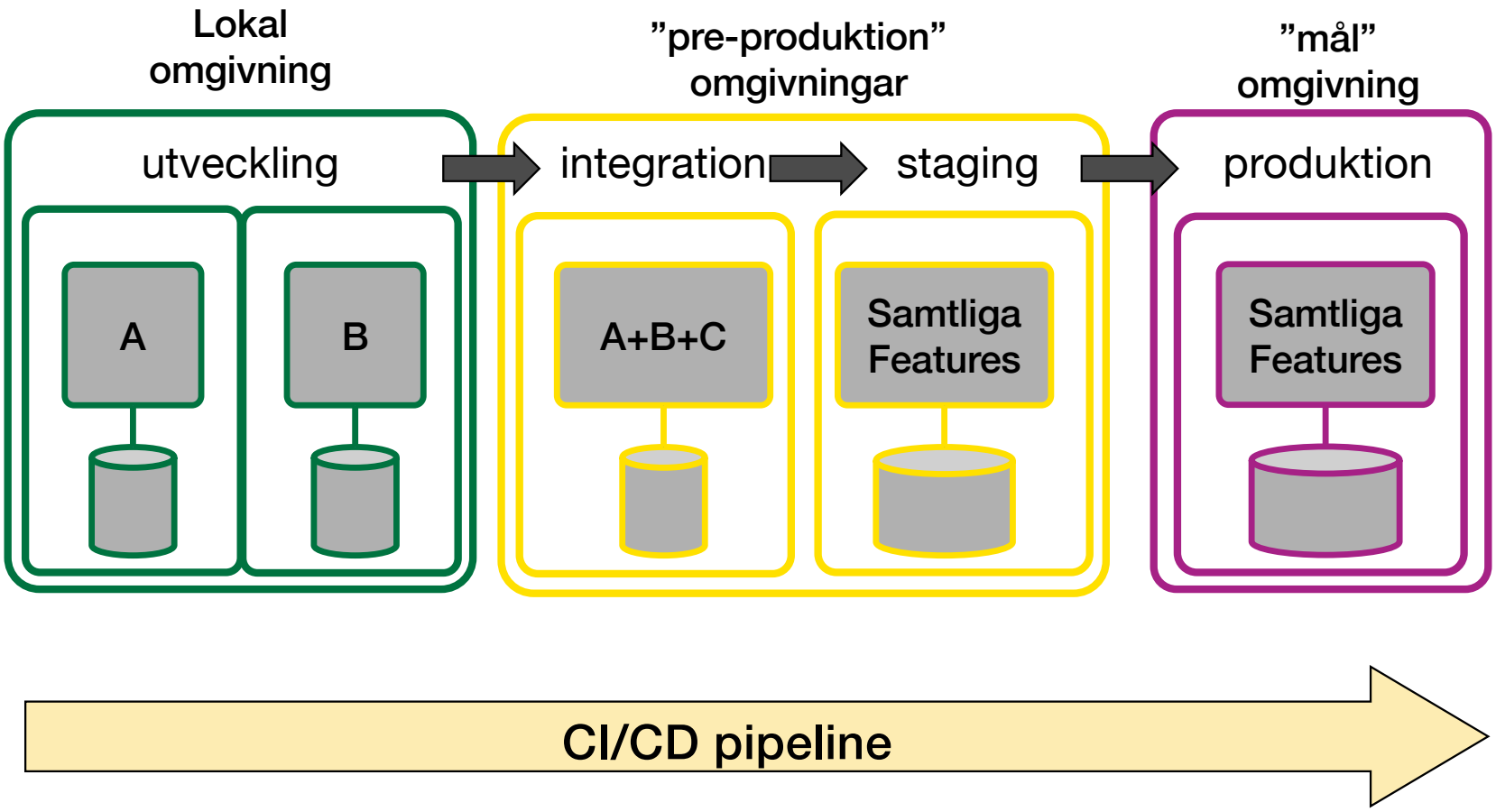
API:er och komposerbarhet är nödvändigt men inte tillräckligt!

**Hur kan vi skapa en arkitektur som underlättar för utvecklare att publicera tjänster – sina API:er – så att deras kunder kan använda dem?**

# Omgivningar



# Omgivningar



# Generellt scenario

| Portion of Scenario | Description | Possible Values |
|---|---|---|
| Source | The trigger for the deployment | End user, developer, system administrator, operations personnel, component marketplace, product owner. |
| Stimulus | What causes the trigger | A new element is available to be deployed. This is typically a request to replace a software element with a new version (e.g., fix a defect, apply a security patch, upgrade to the latest release of a component or framework, upgrade to the latest version of an internally produced element).<br>A new element is approved for incorporation.<br>An existing element/set of elements needs to be rolled back. |
| Artifacts | What is to be changed | Specific components or modules, the system's platform, its user interface, its environment, or another system with which it interoperates. Thus the artifact might be a single software element, multiple software elements, or the entire system. |
| Environment | Where the artifacts are being deployed | Integration environment.<br>Staging environment.<br>Production environment. |
| Response | What should happen | Full deployment.<br>Subset deployed to a specified portion of: users, virtual machines (VMs), containers, servers, platforms.<br>Monitor the new components.<br>Roll back a previous deployment. |
| Response measure | How effective is the deployment, in terms of cost, time, or process effectiveness | Cost in terms of:<br>▪ Number, size, complexity of affected artifacts<br>▪ Average/worst-case effort<br>▪ Elapsed time<br>▪ Money (direct outlay or opportunity cost)<br>▪ New defects introduced<br>Extent to which this deployment/rollback affects other functions or quality attributes.<br>Number of failed deployments.<br>Repeatability of the process.<br>Traceability of the process. |

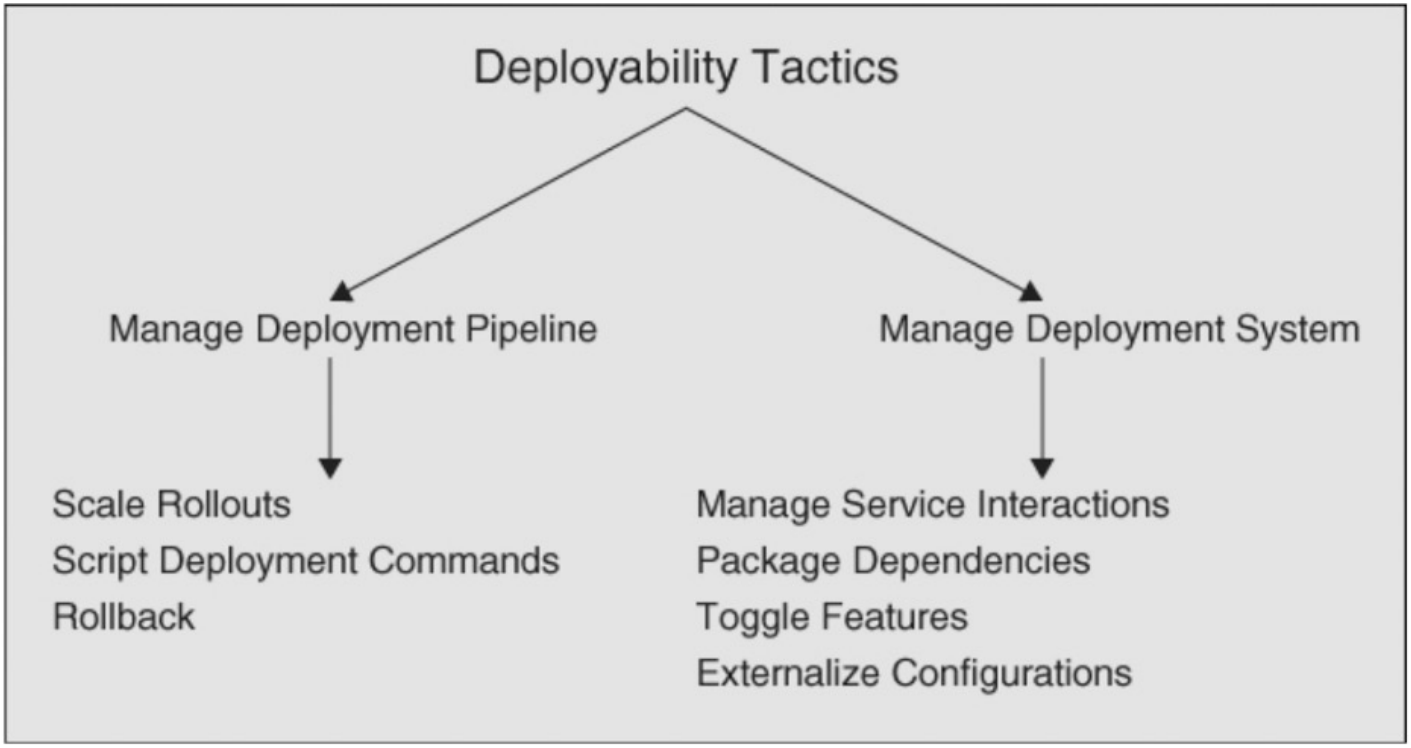
# Design för driftssättning

Upplösning (Granularity). En distribution kan vara hela systemet eller delar av ett system. Om arkitekturen skapar förutsättningar för mer finkorning upplösning i distributionen kan vissa risker reduceras.

Kontrollerbar. Arkitekturen skall gör det möjlighet att distribuera med olika upplösningar, övervaka de delar som distribueras, och ”rulla tillbaka” (rollback) driftssättningar som inte fungerar.

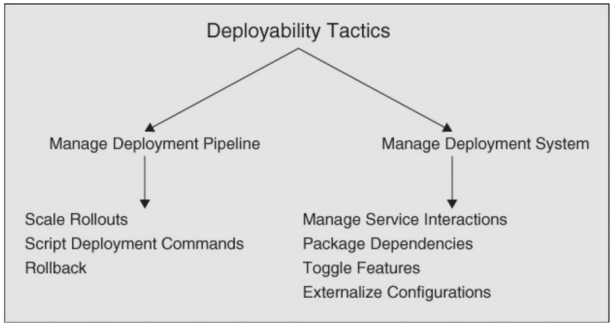
Effektiv. Arkitekturen skall stödja snabb distribution & rollback med rimliga insatser.

# Taktiker


Deployability Tactics
Manage Deployment Pipeline
Manage Deployment System
Scale Rollouts
Script Deployment Commands
Rollback
Manage Service Interactions
Package Dependencies
Toggle Features
Externalize Configurations

# Taktiker

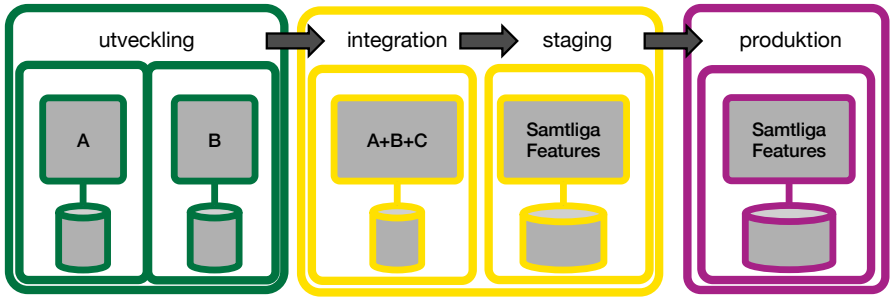
## Manage deployment pipeline




Designbeslut gällande distributionsinfrastrukturen.

**Scale rollouts** Distribuera till delar av användarna. Begränsar effekterna av eventuella felaktigheter och underlättar monitorering och rollback.

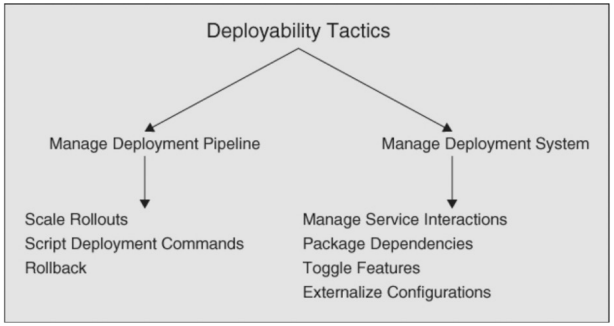
**Roll back**. Om en distribution har defekter behöver den "återställas" till sitt tidigare tillstånd. Eftersom distributioner kan omfatta flera uppdateringar av flera tjänster är ”återställningsmekanismen” komplex och bör automatiseras.




**Script deployment commands.**
De steg som ska utföras vid en distribution bör ”skriptas”.

# Taktiker

## Manage deployment System




Designbeslut gällande distributionsinfrastrukturen.

**Package dependencies.** Element paketeras som ”self-contained”, dvs tillsammans med med sina ”interna beroenden”. Cohesion + Coupling!

**Manage service interactions.** Distribution av tjänster med externa beroenden måste hanteras separat för att undvika inkompatibiliteter.

**Feature Toggle.** En ”brytare" som används för nya funktioner, så att en funktion kan automatiskt deaktiveras vid körning, utan att en ny distribution behöver ske. Minska konsekvenser av felaktiga features.

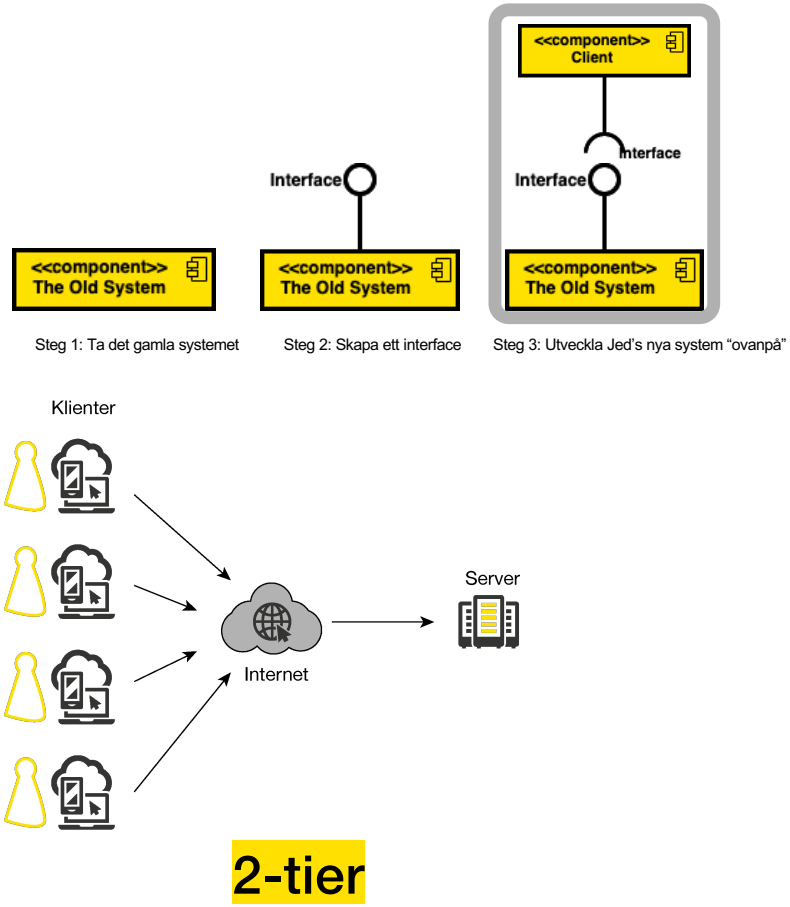
**Externalize configurations.** Systemet bör inte ha några ”hårdkodade” konfigurationer. Det begränsar möjligheten att flytta det från en miljö till en annan (t.ex. från integration till produktion).

# Mönster och deployment

Mikrotjänster

Monoliter och modulära monoliter

N-tier distributioner

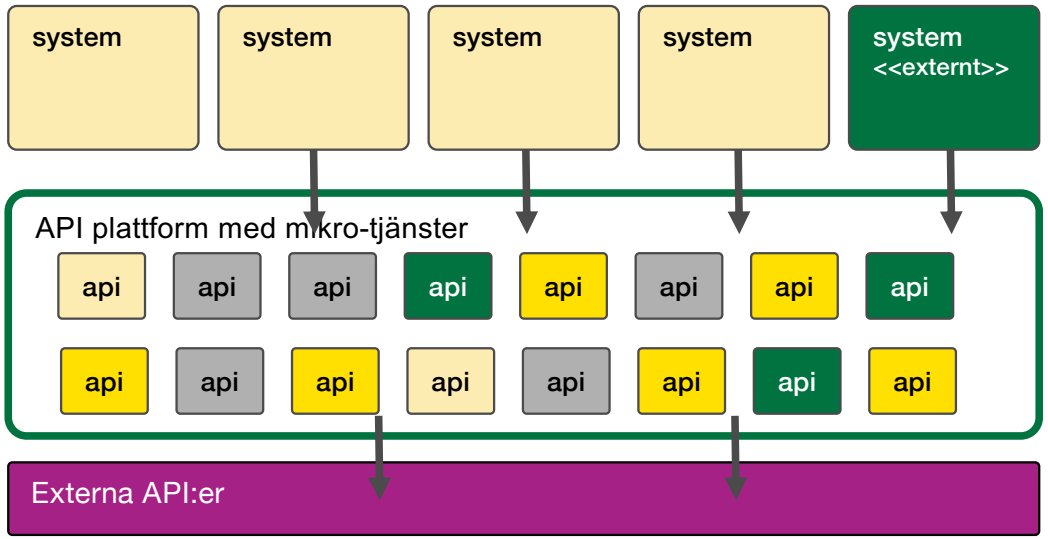


# Från applikationer till system

Mikrotjänster integreras för att skapa ett system

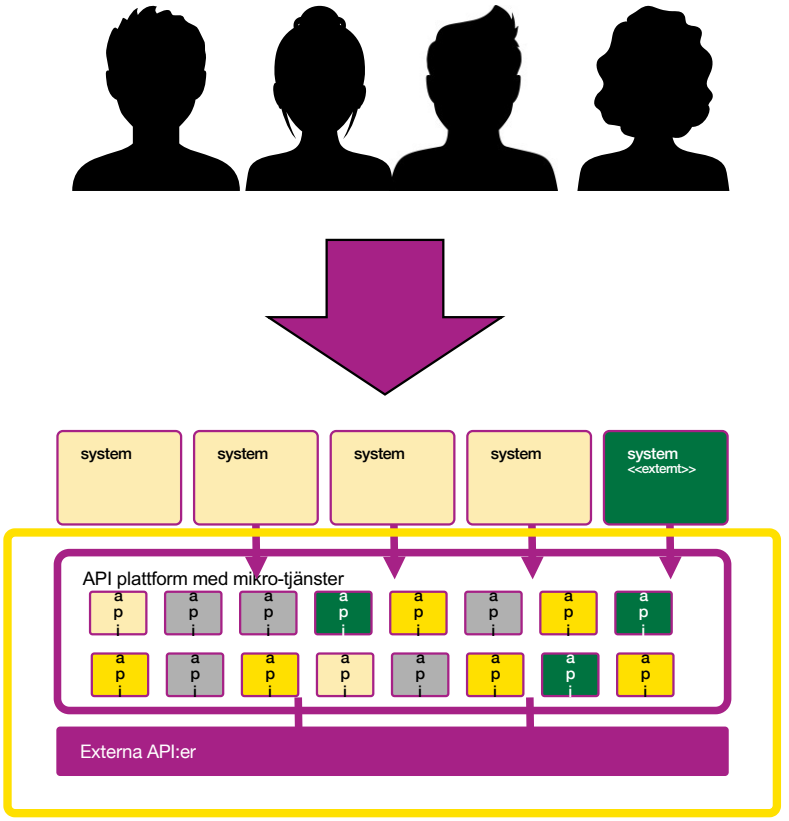
Mikrotjänsterna är separata oberoende komponenter

Integreras via plattformar eller genom en API-gateway



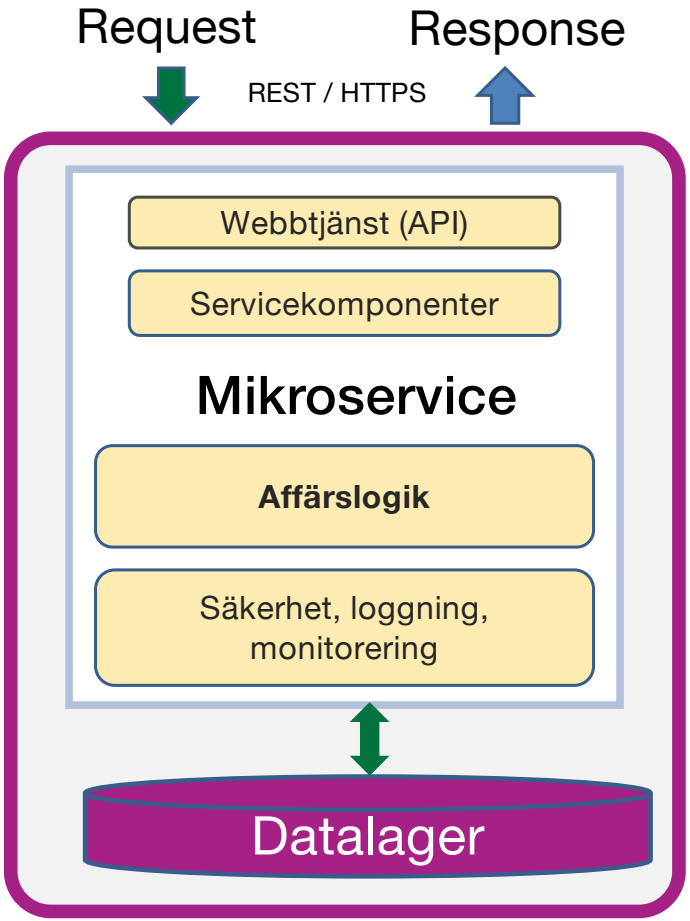
# En intern arkitektur

- Inte uppenbart för användarna om systemet är baserat på mikrotjänster
- En decentraliserad arkitektur innebär inte en “fragmenterad” användarupplevelse utan systemet skapar en sammanhållen upplevelse
- Krävs för snabb prestanda, hög tillgänglighet, extrema belastningar



# Mikrotjänster

## Konceptuell modell



# Skapa en mikrotjänst

Liten enhet med funktionalitet

”Self-contained” och oberoende “teknikstack”

Separata datalager

Synkronisering med andra tjänster efter behov via ett persistenslager

Anropas via API med Request / Response

- Vanligen: REST, HTTP, JSON
- “Self-contained”
  - Inre funktioner som inte exponeras externt
  - Utvecklare kan “fritt” välja tekniska komponenter

# Sätta ihop ett system

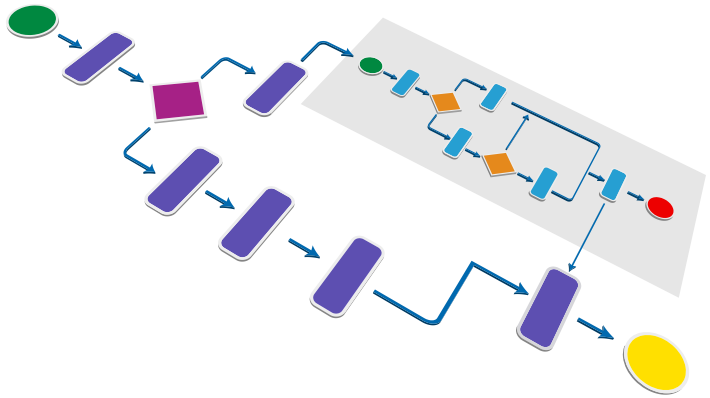
**Varje mikrotjänst utför en liten begränsad uppgift – är inte ett fristående system!**

Orkestrera tjänster

Flera mikrotjänster krävs för att utföra en komplex uppgift

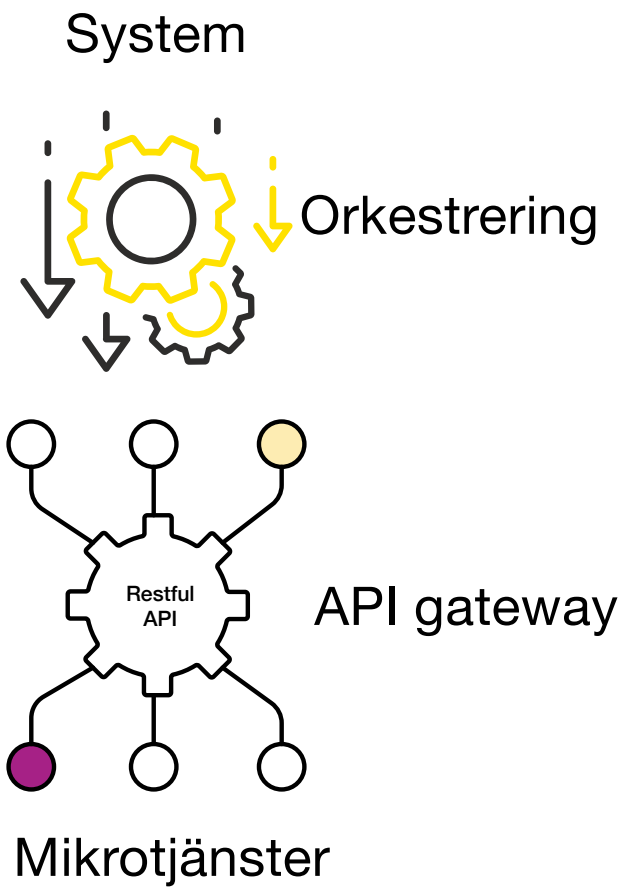
Säkerhet

Lastbalansering



# API Gateway

- Förfrågningar till en mikrotjänst sker vanligtvis inte direkt!
- Det behövs ett lager som hanterar åtkomst till mikrotjänster för att möjliggöra en flexibel distribution.
- En API-gateway är en startpunkt för förfrågningar.
- Utvecklare behöver inte känna till den “fysiska adressen” för varje tjänst (ingenting ska vara hårdkodat).
- Hantera autentisering, protokollkonverteringar, kommunikation, routing, lastbalansering mm.



# Andra mönster

Tillgänglighet

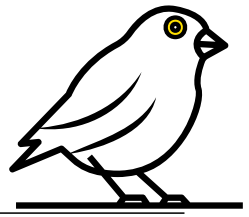
Blue – Green - Med blå-grön driftssättning skapas **n** nya instanser av tjänsten A, de "gröna" instanserna. När **n** instanser driftsatts ändras informationen så att den nya versionen pekas ut.

Rullande uppgradering (Rolling Upgrade) – En rullande uppgradering ersätter instanser av tjänst A med instanser av den nya versionen en efter en.

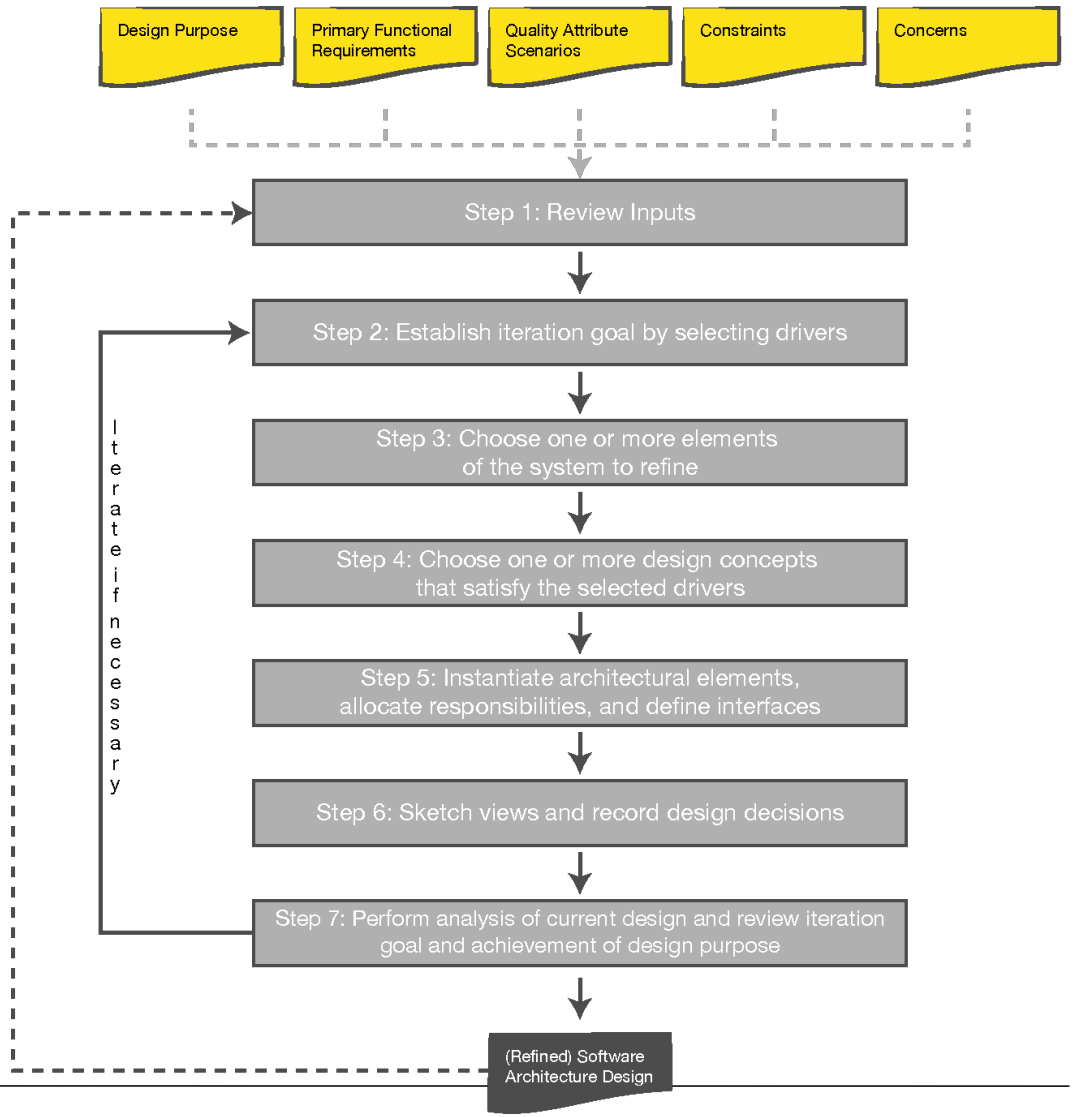
Canary testning (kanariefågel) – Innan en ny version av en tjänst rullas ut så testas den i full drift i produktionsmiljön, men endast för en mindre del av användarna.

A/B testning

Tillförlitlighet



# ADD


Design Purpose
Primary Functional Requirements
Quality Attribute Scenarios
Constraints
Concerns
Step 1: Review Inputs
Step 2: Establish iteration goal by selecting drivers
Step 3: Choose one or more elements of the system to refine
Step 4: Choose one or more design concepts that satisfy the selected drivers
Step 5: Instantiate architectural elements, allocate responsibilities, and define interfaces
Step 6: Sketch views and record design decisions
Step 7: Perform analysis of current design and review iteration goal and achievement of design purpose
Iterate if necessary
(Refined) Software Architecture Design

# Dagens föreläsning

API:er

”composability”

Business agility

Open API

CD/CI

omgivningar

”deployability”

Mönster & taktiker

“Composability,
Pieces fit, systems align,
Deploy with a breeze.”

— ChatGPT 4o

# 2DT902 Mjukvaruarkitekturer

Lnu.se