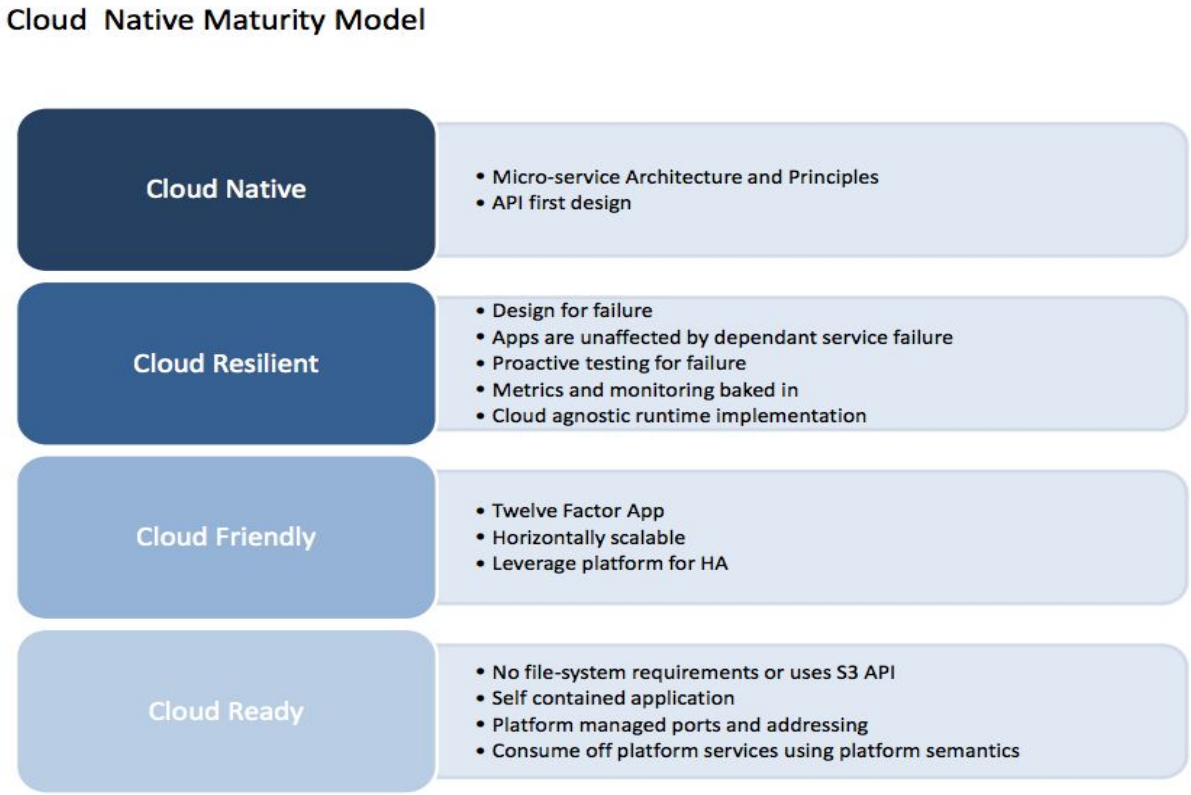
The Cloud Native Journey

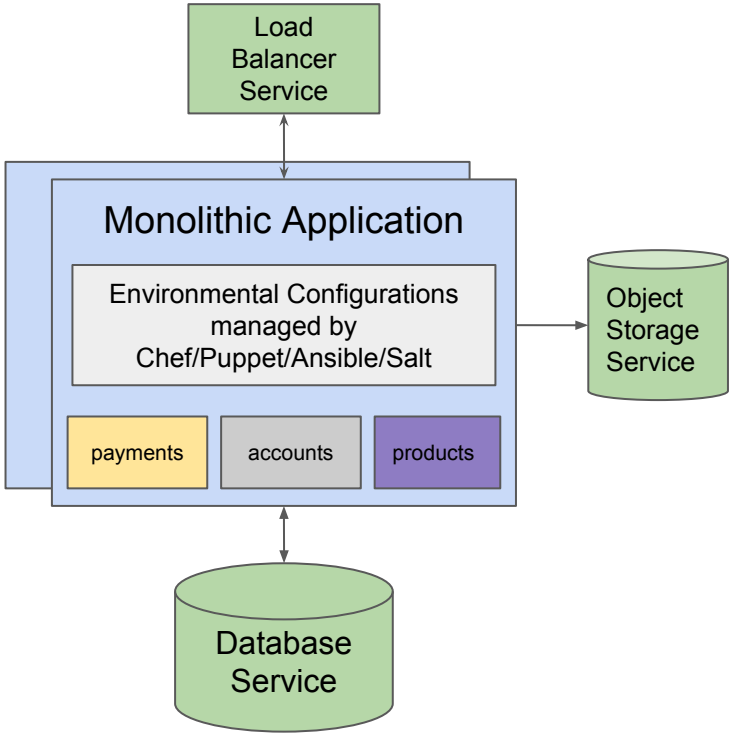
Evolution to a Cloud Native Application

Phases of the Cloud Native Journey



Credits to JP Morgan Chase

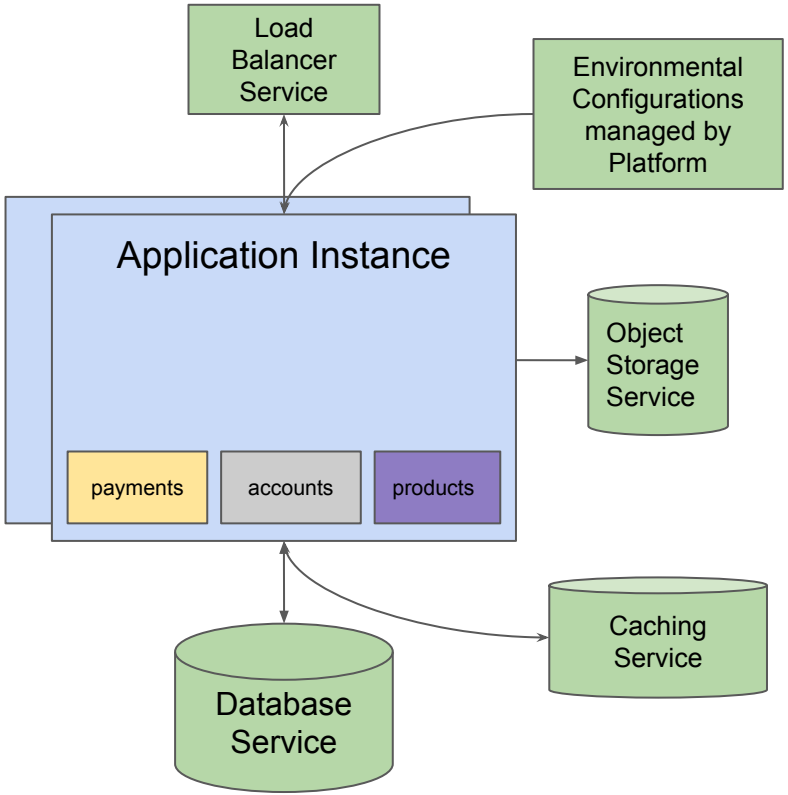
Cloud Ready



Cloud Ready Characteristics:

- Uses a HA shared storage like S3 or OpenStack Swift
- Leverages Platform Services, here it is:
  - Load Balancer
  - Object Storage
  - Database
- Application and dependencies deployed as a single unit
- Platform manages network ports accessible from the outside world
- Application Recovery by Re-Deploying
  - configuration tool used for consistent configurations
  - session state might be lost/service interruption

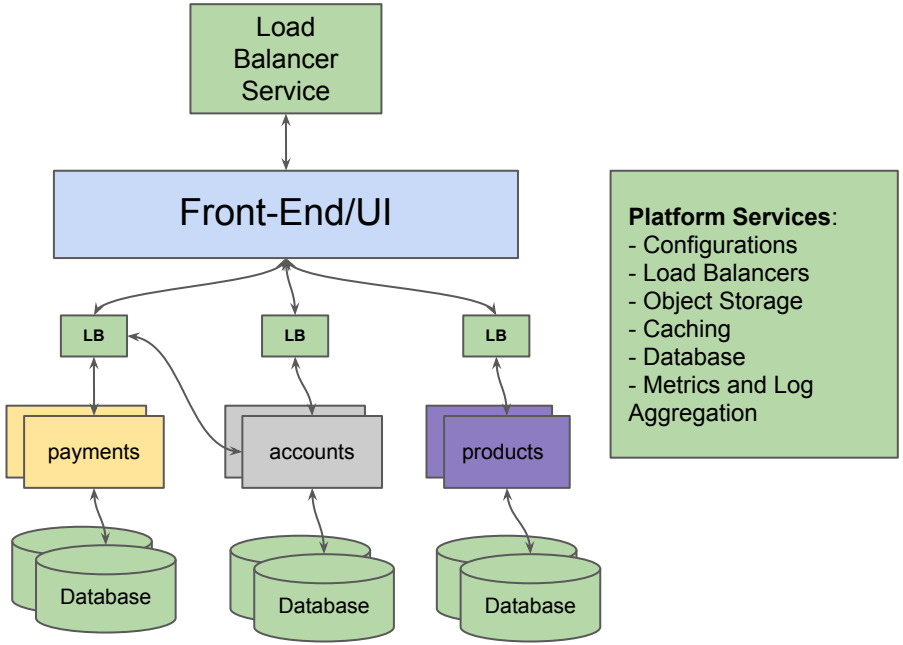
Cloud Friendly



Cloud Friendly Characteristics:

- Moving towards 12-Factor:
  - Platform manages **Environment Configurations**
  - Platform manages **Backend Services**
- Horizontal Scaling:
  - Stateful Data held in Caching Service or database
  - Application Instances can be terminated without loss of service
- Platform now responsible for HA of Backend Services
- Application can now transition to more resilient architecture

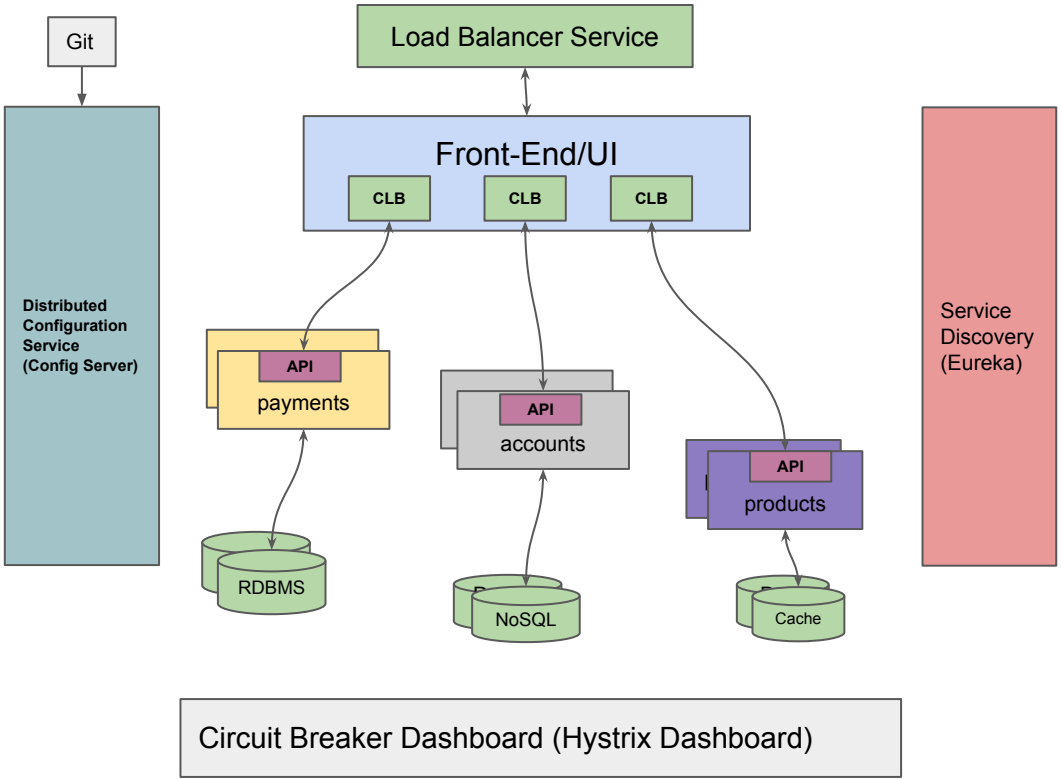
Cloud Resilient



Cloud Resilient Characteristics:

- Design for Failure:
  - Microservices Architecture
  - HA services
  - Services leveraging HA **Platform Services**
- Dependent Service Failure:
  - Each service is now an HA service
  - Service failures can be serviced with pre-determined or cached content
  - Allows for smart error handling if a service does fail
  - Makes the use of Netflix OSS/Spring Cloud Services possible
- Testing is easier on each service; service is decoupled from other services
- Metrics and Log Aggregation is leveraged from the **Platform Services**
  - Easier analysis and visualization of a given service is more impactful and insightful to application as a whole
- Platform inherently abstracts IAAS APIs making the application Cloud Agnostic

Cloud Native



Cloud Native Characteristics - Adoption of Microservice Architectural principles:

- Services register and discover other services using a light weight service discovery tool like Netflix Eureka.
- Services obtain configuration at startup or during runtime from a config server.
- Services employ anti fragility practices using circuit breaker patterns with Netflix Hystrix
- Services use client side load balancers (CLB) to load balance requests amongst different instances of the same downstream service using Netflix Ribbon.

Services are built with an API first design in mind using a light weight OSS API builder like Spring REST Docs.