Pattern: Service deployment platform

Context

You have applied the Microservice architecture pattern (/patterns/microservices.html) and architected your system as a set of services. Each service is deployed as a set of service instances for throughput and availability.

Problem

How are services packaged and deployed?

Forces

- Services are written using a variety of languages, frameworks, and framework versions
- Each service consists of multiple service instances for throughput and availability
- Service must be independently deployable and scalable
- Service instances need to be isolated from one another
- You need to be able to quickly build and deploy a service
- You need to be able to constrain the resources (CPU and memory) consumed by a service
- You need to monitor the behavior of each service instance
- You want deployment to reliable
- You must deploy the application as cost-effectively as possible

Solution

Use a deployment platform, which is automated infrastructure for application deployment. It provides a service abstraction, which is a named, set of highly available (e.g. load balanced) service instances.

Examples

- Docker orchestration frameworks including Docker swarm mode (https://docs.docker.com/engine/swarm/) and Kubernetes (http://kubernetes.io/)
- Serverless platforms (serverless-deployment.html) such as AWS Lambda
- PaaS including Cloud Foundry (https://www.cloudfoundry.org/) and AWS Elastic Beanstalk (https://aws.amazon.com/elasticbeanstalk/)

Related patterns

- Some deployment platforms provide a Service Registry (../service-registry.html) and Server-Side discovery (../server-side-discovery.html)
- Internally, a deployment platform might use use containers (../deployment/service-per-container.html) or virtual machines (../deployment/service-per-vm.html) to deploy a service. Docker orchestration frameworks are, of course, explicitly container-based (../deployment/service-per-container.html)