

Deployment Strategies

CSSE6400

Richard Thomas

May 6, 2024

Definition 1. Deployment Strategy

How a software system is made available to clients.

Deployment Strategies

- Branching Strategies
- Recreate Deployment
- Rolling Deployment
- Blue/Green Deployment
- Canary Deployment
- A/B Deployment
- Shadow Deployment

Definition 2. Branching

Copying the trunk to allow separate and parallel development.

Branching Strategies

- GitHub Flow
- GitLab Flow
- Release Branches

GitHub Flow *[Haddad, 2022]*

- Main is always deployable
- Create branch
- Make changes
- Create pull request
- Resolve issues
- Merge pull request
- Delete branch



GitLab Flow *[PRACE,]*

- Supports deployment windows
 - Merge to production
 - Deploy when allowed
- Production branch
 - Plus alpha, beta, ...
- Still have
 - Feature branches
 - Pull requests



Release Branches ^[PRACE,]

- Supports multiple versions of system
- Feature development in main
- Released versions are branches
- Bug fixes in main
 - Cherry-pick into branches



Recreate Deployment *[Tremel, 2017]*



Recreate Deployment

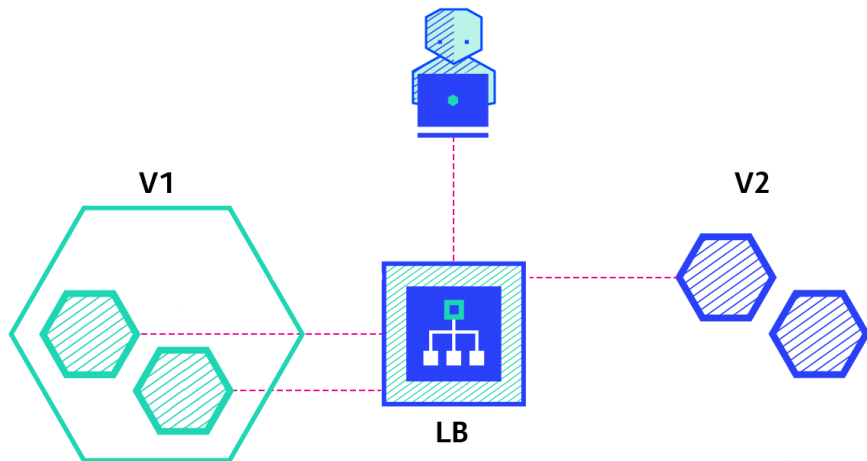
Pros

- Easy
- Renewed state
 - App reinitialised
 - Persistent storage consistent with system version

Cons

- Downtime

Rolling Deployment *[Tremel, 2017]*



Rolling Deployment

Pros

- Fairly easy
- Slow release of new version
 - Observe issues
 - Rollback
- Stateful instances can finish gracefully
 - Instance is killed when inactive

Cons

- Time
- Support multiple APIs
- Support different versions of persistent data structure
- No control over traffic to different versions

Blue-Green Deployment *[Tremel, 2017]*



Blue-Green Deployment

Pros

- Instant release of new version
- Fast rollback if necessary
- Only one version 'live' at any time
 - No versioning conflicts

Cons

- Expensive
 - Double the infrastructure
- Stateful instance version switch difficult
 - Can't kill instance in middle of a transaction

Canary Deployment *[Tremel, 2017]*



Canary Deployment

Pros

- New version released to subset of users
- Can monitor performance and error rates
- Easy and fast rollback

Cons

- Slow
- Implies poor testing

A/B Deployment *[Tremel, 2017]*



A/B Deployment

Pros

- Multiple versions run in parallel
- Full control over traffic distribution

Cons

- Needs intelligent load balancer
- Debugging a version is difficult
 - Need good logs & tools

Shadow Deployment *[Tremel, 2017]*



Shadow Deployment

Pros

- Performance testing with production traffic
- No impact on users

Cons

- Expensive
 - Double the infrastructure
- Complex to setup
 - Need mocks for external services

Deployment Strategy Options

- Staging or beta testing
 - Recreate or Rolling
- Production (Live)
 - Rolling or Blue/Green
- Uncertain of system stability
 - Canary
- Evaluation
 - A/B or Shadow

Deployment Considerations *[Tremel, 2017]*

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■ □ □	■ ■ ■	■ ■ ■	□ □ □
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■ □ □	■ ■ ■	■ □ □	■ □ □
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■ ■ ■	□ □ □	■ ■ □	■ ■ □
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■ □ □	■ □ □	■ □ □	■ ■ □
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■ □ □	■ □ □	■ □ □	■ ■ ■
SHADOW version B receives real world traffic alongside version A and doesn't	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

References

[Haddad, 2022] Haddad, R. (2022).

What are the best git branching strategies.

<https://www.flagship.io/git-branching-strategies/>.

[PRACE,] PRACE.

Introduction to gitlab flow.

https://repository.prace-ri.eu/git/help/topics/gitlab_flow.md.

[Tremel, 2017] Tremel, E. (2017).

Six strategies for application deployment.

<https://thenewstack.io/deployment-strategies/>.