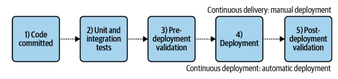
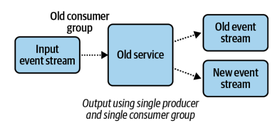
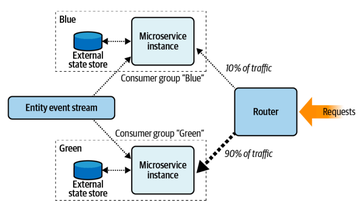
- Adam Bellemare - Building Event-Driven Microservices
  - 
  - 16. Deploying Event-Driven Microservices
    - Principles of Microservice Deployment
      - Give teams deployment autonomy
      - Implement a standardized deployment process
      - Provide necessary supportive tooling
      - Consider event stream reprocessing impacts
      - Adhere to service-level agreements (SLAs)
      - Minimize dependent service changes
      - Negotiate breaking changes with downstream consumers
    - Architectural Components of Microservice Deployment
      - Continuous Integration, Delivery, and Deployment Systems
        - the system used to build and deploy the code
        - Continuous integration (CI)
          - is the practice of automating the integration of code changes from multiple contributors into a single software project
        - Continuous delivery
          - the practice of keeping your codebase deployable
        - Continuous deployment
          - the automated deployment of the build
        - 

        - A CI pipeline showcasing the difference between continuous delivery and continuous deployment
      - Container Management Systems (CMS) and Commodity Hardware
        - the compute resources used by the microservices
    - The Basic Full-Stop Deployment Pattern
      - 1. Commit code
      - 2. Execute automated unit and integration tests
      - 3. Run predeployment validation tests
        - Event stream validation
        - Schema validation
      - 4. Deployment
        - Stop instances and perform any clean-up before deploying
        - Deploy
      - 5. Run post-deployment validation tests
    - The Rolling Update Pattern
      - No breaking changes to any state stores
      - No breaking changes to the internal microservice topology
        - particularly relevant for implementa-tions using lightweight frameworks
      - No breaking changes to internal event schemas
      - New fields have been added to the input events and need to be reflected in the business logic
      - New input streams are to be consumed
      - Bugs need to be fixed but don't require reprocessing
      - Inadvertently altering the internal microservice topology is one of the most common mistakes
        - It is a breaking change!
    - The Breaking Schema Change Pattern
      - ENTITY schema changes
        - Are more complex
        - will require reprocessing the necessary source data for the producer
        - 

        - Breaking schema change producer options for re-creating events with new schema
      - NON-ENTITY schema changes
        - may not require reprocesiing
      - Two options
        - Eventual Migration via Two Event Streams
          - The producer write events with both the old and new format, each to its respective stream
          - assumptions
            - Events can be produced to both the old and new streams.
            - Eventual migration will not cause downstream inconsistencies
          - the main risk
            - the migration is never finished (similar-yet-different data streams remain in use indefinitely)
        - Synchronized Migration to the New Event Stream
          - a rare case in practice
          - The producer to create events strictly with the new format
          - assumptions
            - The event definition change is significant enough that the old format is no longer usable
            - Migration must happen synchro-nously to eliminate downstream inconsistencies
          - the main risk
            - consumers may fail in their migration to the new event stream
              - but be unable to gracefully fall back to the old source of data
    - The Blue-Green Deployment Pattern
      - 

      - Blue-green deployment pattern
      - work well when
        - consume from event streams
        - events are produced only due to request-response activity
      - do not work when
        - the microservice produces events to an output stream in reaction to an input event stream
          - two microservices will overwrite or duplicate each other's results
          - Use either the rolling update pattern or the basic full-stop deployment pattern instead