Using Automation for Generic Mitigations in Production

May 20, 2021

Presenter: Leonid Belkind, Co-Founder & CTO, StackPulse
Agenda

- Generic Mitigations
- Automation: Playbooks-as-Code
- Bringing it all together...
  - Samples
  - Architecture Blueprint
  - Main principles
Typical timeline of an outage

1. **Monitoring**
2. **Alert**
3. **Enrichment / Hydration**
4. **Triage**
5. **Generic Mitigation**
6. **Root Cause Analysis**
7. **Fix**
8. **Deploy a Fix**

Reduction in outage time
Gain in SLO
So, what kind of Generic Mitigations do you have in mind?

- Rollback (Business Logic, Configuration, Data)
  The ability to safely return to a working state. This might sound simple, rolling back a deployment of a single component, but, actually, performing this in a multi-component environment, with dependencies and evolving data schemas is not straight forward at all.

- Upsize / Downsize
  The ability to increase / decrease amount of replicas of a certain component, while continuously handling the production traffic. Ability to do that controlling the system externally without invoking infrastructure and application specialists to perform dangerous changes in production.

- Drain and Flip Traffic
  The ability to gradually drain the connections from a specific instance / site / cluster (experiencing errors) and transfer them to another one. Doing so safely, without involving ad-hoc operations in production.
Is that it? .......... Of course not

• Quarantine
  After identifying a “bad” instance in a cluster, remove it from rotation ensuring that the other instances continue handling traffic without impact on the users. Then, investigate the root cause of the problem.

• Block List
  Block a specific user / account / session / external source of problematic requests to make sure that it doesn’t impact the overall delivery of your service to the rest. Potentially, add specific quotas / guardrails on this particular source rather than just blocking it.

• Disable a Noisy Neighbour
  Identify the source of “noisy neighbour” (for example, in a database, sending long queries that require too much resources) and terminate the queries / sessions that impact others.
“Generic Mitigations” is a practice of improving SLOs and returning the service to operational state faster, without compromising on Root Cause Analysis and good software engineering practices.

Building Generic Mitigations and testing them (to build confidence to apply them to production) is a very important aspect of becoming proficient in building resilient systems.
Playbooks
(a.k.a. Runbooks)

A playbook includes process workflows, standard operating procedures, and cultural values that shape a consistent response — the play.

Enable consistent and prompt responses to failure scenarios by documenting the investigation process in playbooks. Playbooks are the predefined steps to perform to identify and resolve an issue.
Sample Playbooks

K8S PoD Restarting - Operational Runbook

This playbook describes the operations required to troubleshoot the data about K8S PoD being restarted in a production environment.

1. Understand the Cluster, Node, Namespace, Pod, and Container Data
   The user needs to understand what has restarted in which namespace, namespace within the Kubernetes cluster. Start your investigation by looking at the:

   - **Cluster**: Check the overall health of the cluster and the pods within it.
   - **Node**: Check the nodes where the pods are running.
   - **Namespace**: Check the namespaces that contain the pods.
   - **Pod**: Check the specific pod that has restarted.
   - **Container**: Check the specific container that has restarted.

2. Understanding the Logs
   Looking at the information available, we need to identify the instances of the event that is described. This is a crucial step towards identifying the root cause. It has a significant impact on our analysis and is focused on detectability in large teams.

   - **CloudTrail Logs**: Check for any failing or successful API calls in the CloudTrail logs.
   - **Container Logs**: Check the logs for the container that has restarted.
   - **Kubernetes Events**: Check for any system events or errors.
   - **Network Logs**: Check for any network errors or failures.
   - **Application Logs**: Check for any application-specific errors or failures.

3. Pull the latest logs from the crashed container to understand the reason
   The logs can be pulled from the container itself, but depending on the service and its configuration, the logs may not be accessible. Therefore, we recommend using the **kubectl** command to access the logs directly from the container.

2.1 Pulling logs from a container

   - **Get the **Pod** name and **Container** name**: First, get the **Pod** name and the **Container** name that has restarted.
   - **Pull the logs using the following command**:

4.2 Pulling logs directly from a Kubernetes Cluster

   Using the information above, we can directly access the pod to the relevant environment. For access to K8S cluster, please follow the following command:

   ```
   kubectl logs <pod-name> <container-name>
   ```

   Example:

   ```
   kubectl logs <pod-name> <container-name>
   ```

   - **Access to the pod**: Once the logs are retrieved, you can use **kubectl exec** to further inspect the logs.
   - **Inspect the logs**: Use **less** or a terminal pager to view the logs.
   - **Analysis of the logs**: Look for any error messages, warnings, or other indicators that can help identify the root cause.

5. Identify root causes

   - **Check for any external dependencies**: Look for any external services that may be affected.
   - **Check the pod’s configuration**: Ensure that the pod’s configuration is correct.
   - **Check the container’s configuration**: Ensure that the container’s configuration is correct.
   - **Check the network configuration**: Ensure that the network configuration is correct.

6. Resolution

   - **Resolve the issue**: Once the root cause is identified, resolve the issue.
   - **Test the resolution**: Test the resolution to ensure that it works as expected.
   - **Document the resolution**: Document the resolution for future reference.

7. Verification

   - **Verify the resolution**: Verify that the resolution has been successful.
   - **Test the environment**: Test the environment to ensure that it is stable.
   - **Update the documentation**: Update the documentation to reflect the changes made.
Playbooks – The essence

Playbooks provide (hopefully) step-by-step guides for human operators to ensure repeatable and consistent response to incident situations.

But why?

If we have repetitive technical step-by-step procedures, isn’t automation the better option to ensure efficiency and consistency?

Enter Playbooks-as-Code
Google Research on Incident Response

Building Blocks

- **Triage**
  - Evaluating the Alert
    - Should I create an incident?
    - Should I escalate to a more severe incident?
    - Should I close it as inactive?
    - Is it local or global?
  - Isolating the Error
    - Was there a spike in errors and latency?
    - Was there a change in demand? (e.g., CPU)
    - What are the anomalous dimensions driving errors?
    - What is the error? (Group-by, log, etc.)
- **Investigate**
  - Determining if It’s Me or If It’s My Dependency’s Issue
    - What are my problematic dependencies?
  - Validating if my Service is Healthy
    - What is my SLA?
    - Am I close to or exceeding my SLAs?
  - Determining Change Around Me
    - What production changes (intra, config, data paths, experiment) could correlate to this?
    - Was there a spike in demand?
- **Mitigate**
  - Mitigating the Issue
    - What mitigation should I take?
    - How confident am I that this is the right mitigation?
  - Validating if my Service is Healthy
    - Did the mitigation fix the issue?

User Journey

* From “Debugging Incidents in Google’s Distributed Systems, Charisma Chan and Beth Cooper, 2020*
What are playbooks-as-code

Playbooks-as-code are deterministic manual operator instructions converted into automation processes

Automate Chat Channels, Notifications

Automate Messaging / Paging to Stakeholders

Automatically Update Incident Based on Enriched Data

Automate Pulling Impact Data from Logs/Context

Automate Diagnostics, Look for Suspicious Signals

Automate Log Queries, Calculate Data on Pulled Logs

Automatically Connect to SCM/CI Pipeline

Automate Remediative Actions / CLI / API / ...
Playbooks-as-Code ➔ The Approach

- Similar to CI/CD Pipelines or Automated Tests, define playbook workflows as high-level code/configuration

- Apply Software Engineering principles to Incident Response Playbooks
  - Break down into modules
  - Handle specific tasks, as a part of Incident handling
  - Think of arguments / parameters and encapsulation allowing re-use
  - Consider sharing between teams in the organization and between organizations
  - Visualize / Troubleshoot / Audit execution
  - Apply SDLC, GitOps, ...

- Have a Playbooks-as-Code orchestrator separate from our application infrastructure
Playbook-as-Code Sample

- name: stackpulse/public/ssh/command
  id: ssh_command
  env:
    USERNAME: "{{ $.params.UserName }}"
    HOSTNAME: "{{ $.params.ServerAddress }}"
    COMMAND: df -k
    PRIVATE_KEY: '>{{ secret "SSH_KEY"}}'

- name: stackpulse/public/slack/message
  id: slack_send_message
  env:
    MESSAGE_TEXT: |
    The filesystems for server {{ $.params.ServerAddress }}
    ...
    {{ $.ssh_command.output }}
    ...
    RECEipients: "alerts"
Typical Architecture for Playbooks-as-Code Orchestration

Production Environments -> Monitoring Systems -> Playbook Orchestrator -> Communication / Collaboration Systems
Summary

- Well-defined, pre-rehearsed and deterministic processes are a MUST to ensure efficient handling of incidents

- A “library” of Generic Mitigations ensures ability to reduce outages

- “Documenting Step-by-Step Directions for Human Operators” is not the way to go. There is a better alternative, as proven by:
  - Automated Testing
  - Automated Integration / Deployment
  - Infrastructure-as-Code
  - ...

- Think of actions taking place during incidents / alerts just as of another aspect of “code” and act accordingly
Thank You!

Questions?
leonid@stackpulse.com