Four Nines of Availability

Part 1 : Micro-services

Hike Follow
Dec 20, 2018 · Unlisted

By Aditya Gupta, Director Engineering, Hike

For any product or service, availability is one of the most important factors and when you are a messaging app it becomes even more important to have highest availability of your services with lowest latency.

> *"Availability is not just sending a response back, it has to be in a specified time."*

We had an interesting journey from where we started to even think of four nines of availability for all our services.

Just to give a glimpse of what it means:

- *Five-nines or 99.999% availability means 5 minutes, 15 seconds or less of downtime in a year*
- *Four nines or 99.99% availability allows 52 minutes, 36 seconds downtime per year*
- *Three nines or 99.9% availability allows 8 hours, 46 minutes downtime per year*

- *Two nines or 99% availability allows 3 days, 15 hours and 40 minutes downtime per year.*
- *One nine or 9% availability allows over 332 days of downtime per year.*

At the first glance, 52 minutes of downtime in a year seems too much but what this means is that you can vary your SLA's for just 8.65 seconds in an entire day (including peak traffic hours)

Why is five, four or even three nines of availability so hard to achieve?

1. *Bugs: Human error still accounts to around 60% of the downtime*
2. *SPOF: Single point of failures leading to system wide downtime*
3. *Hardware failures, Network issues*
4. *Unusual traffic spikes*
5. *Third party or external APIs dependencies*

In order to improve on availability of overall system, you need to work on each of the above points — after all the*"Strength of a chain is as good as strength of its weakest link"*.

So, where do you start if you want to improve on availability?

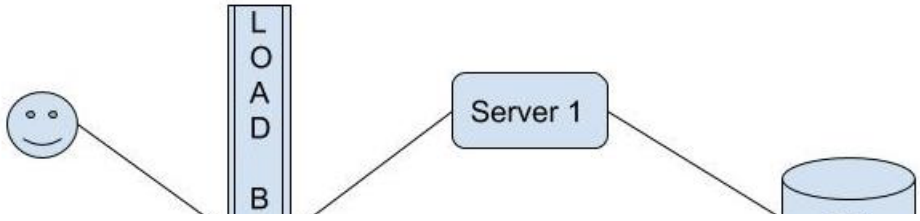
1. Monitoring

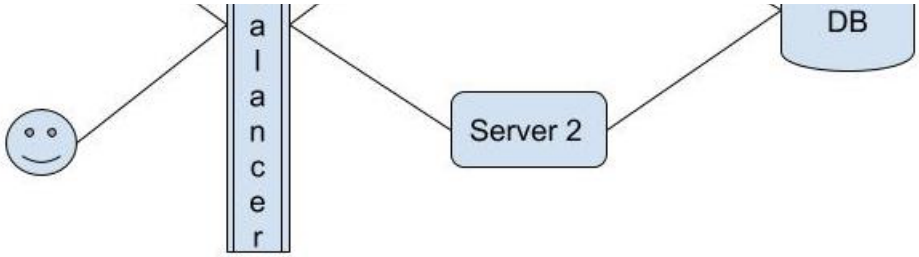
The first and foremost step to improve on anything is to monitor it regularly and have clear alerts whenever it fails

2. Eradicate Single point of failures

Back in 2012 when we started, we had a monolith architecture where all the components were together in a single deployable unit and all the processing was done synchronously.

It was literally something like this.

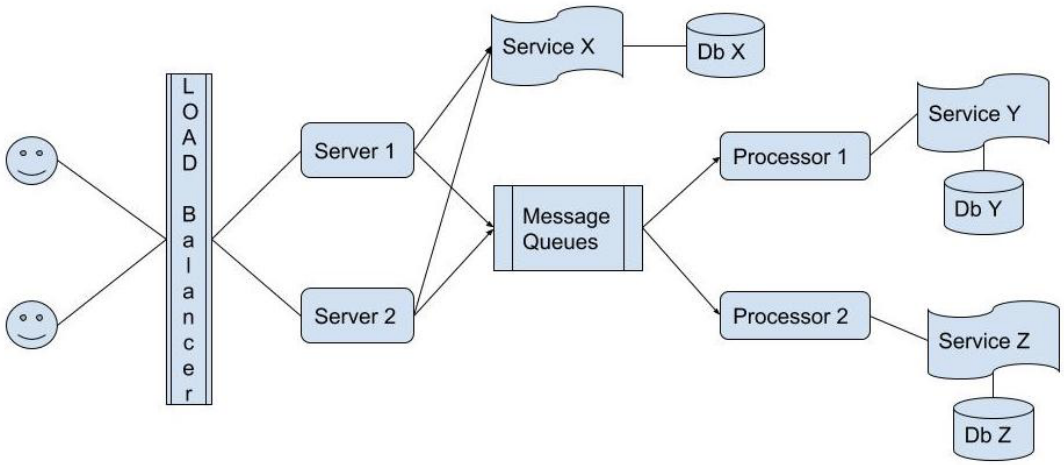




Though it gave us the speed (initially), as it was quick to develop but there were many pitfalls like:

- Single point of failures (SPOF), so if there is an issue in one of the flows, the whole system gets affected.
- Difficult to do fault detection and isolation
- Continuous deployment become an issue
- Application becomes large and too complex to understand

To avoid these pitfalls, we broke down our architecture to a service-oriented one with clear separation of services, which not only helped to eradicate SPOF but also make fault detection and isolation easy.



Now, let's say 'Db Y' is down then only service Y or Processor 1 will get affected and rest of the system will remain healthy. For fault detection and isolation we are using Hystrix for some and an internal tool for other layers.

3. Tackling Bugs

> *"Monitor → Fix → Automate → Repeat"*

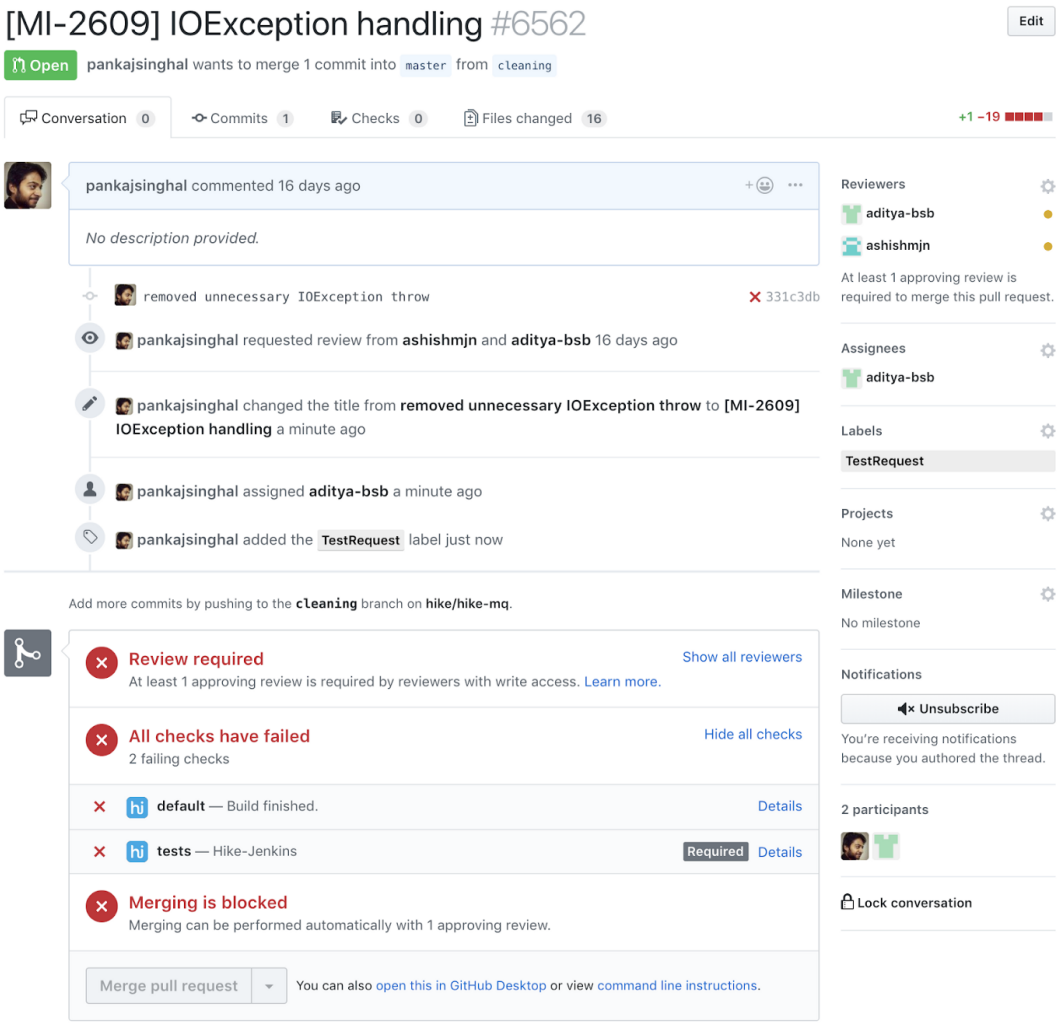
Monitor as many things you can, run your automation test suite as early as possible, make hard checks at every step to stop bad code/configuration from reaching your production environment.

Stop bad code from reaching the production system

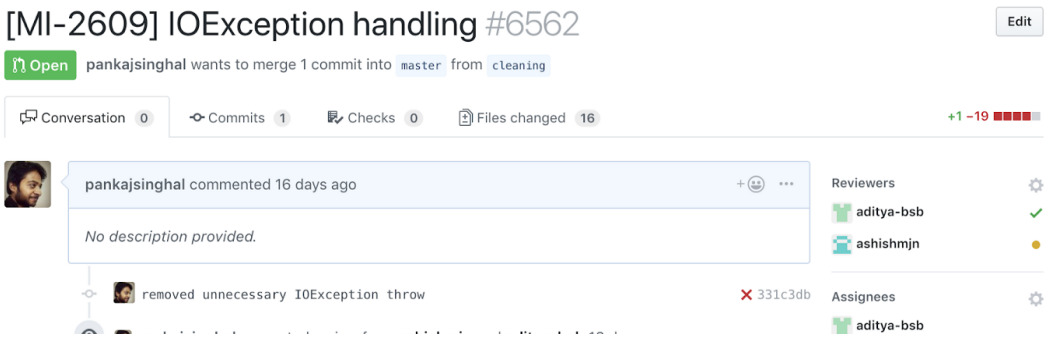
Bugs often reach the production systems because of missing edge condition, developer didn't get it reviewed or just missed to run test cases around it.

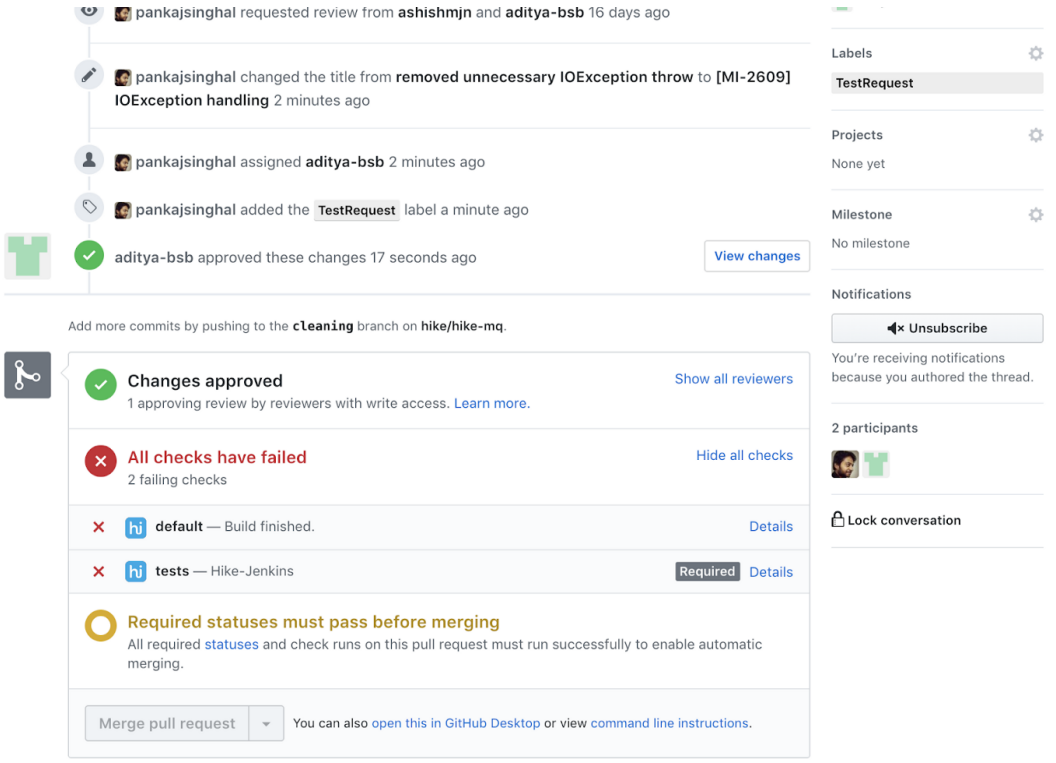
To prevent this, we have put the following mandatory checks on every *'pull request'* raised by a developer.

- All automated test suites must have run successfully
- Should be reviewed by at least one other developer

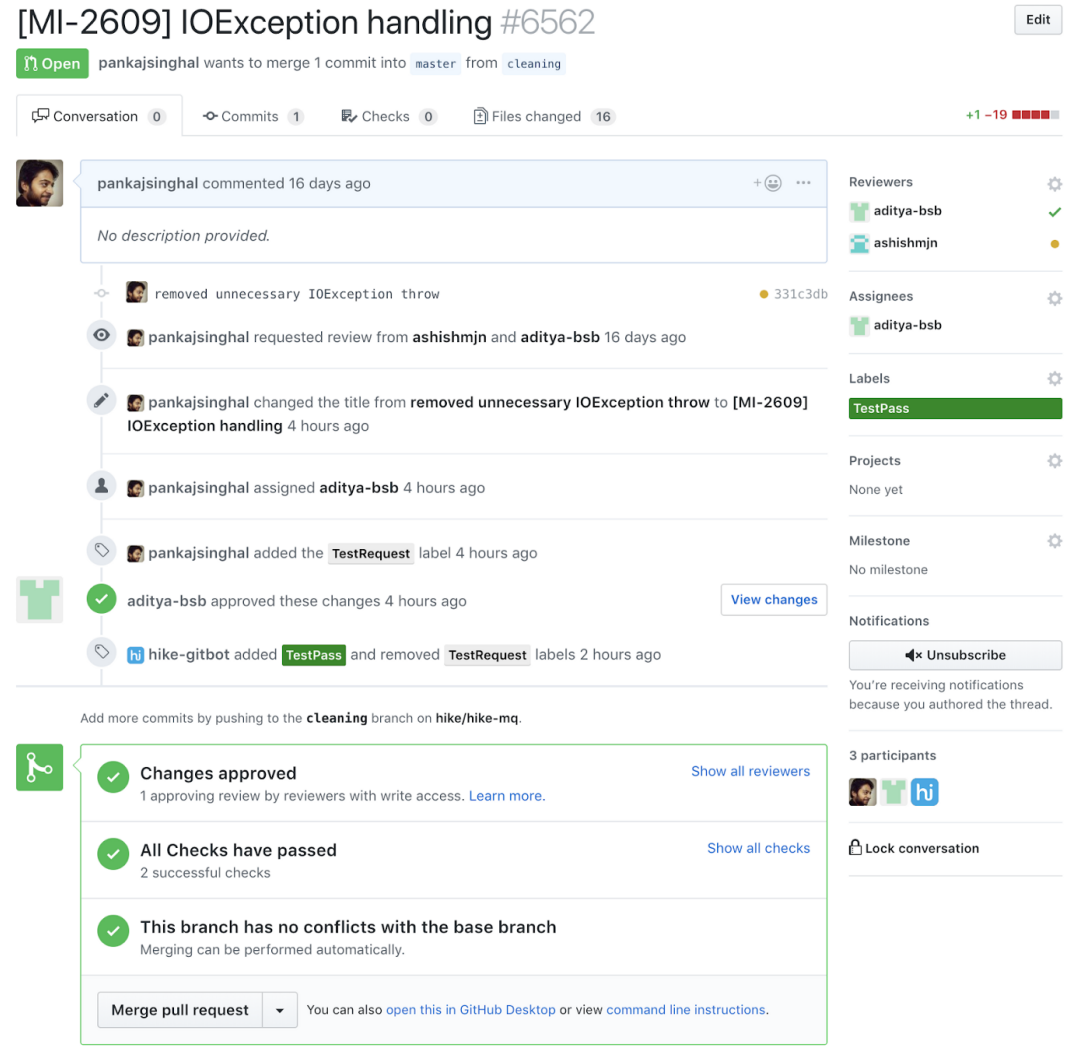


Pull Request (PR) Raised




pankajsinghal requested review from ashishmjn and aditya-bsb 16 days ago
pankajsinghal changed the title from removed unnecessary IOException throw to [MI-2609] IOException handling 2 minutes ago
pankajsinghal assigned aditya-bsb 2 minutes ago
pankajsinghal added the TestRequest label a minute ago
aditya-bsb approved these changes 17 seconds ago
View changes
Add more commits by pushing to the cleaning branch on hike/hike-mq.
Changes approved
1 approving review by reviewers with write access. Learn more.
Show all reviewers
All checks have failed
2 failing checks
Hide all checks
default — Build finished.
Details
tests — Hike-Jenkins
Required
Details
Required statuses must pass before merging
All required statuses and check runs on this pull request must run successfully to enable automatic merging.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Labels
TestRequest
Projects
None yet
Milestone
No milestone
Notifications
Unsubscribe
You're receiving notifications because you authored the thread.
2 participants
Lock conversation

PR Approved

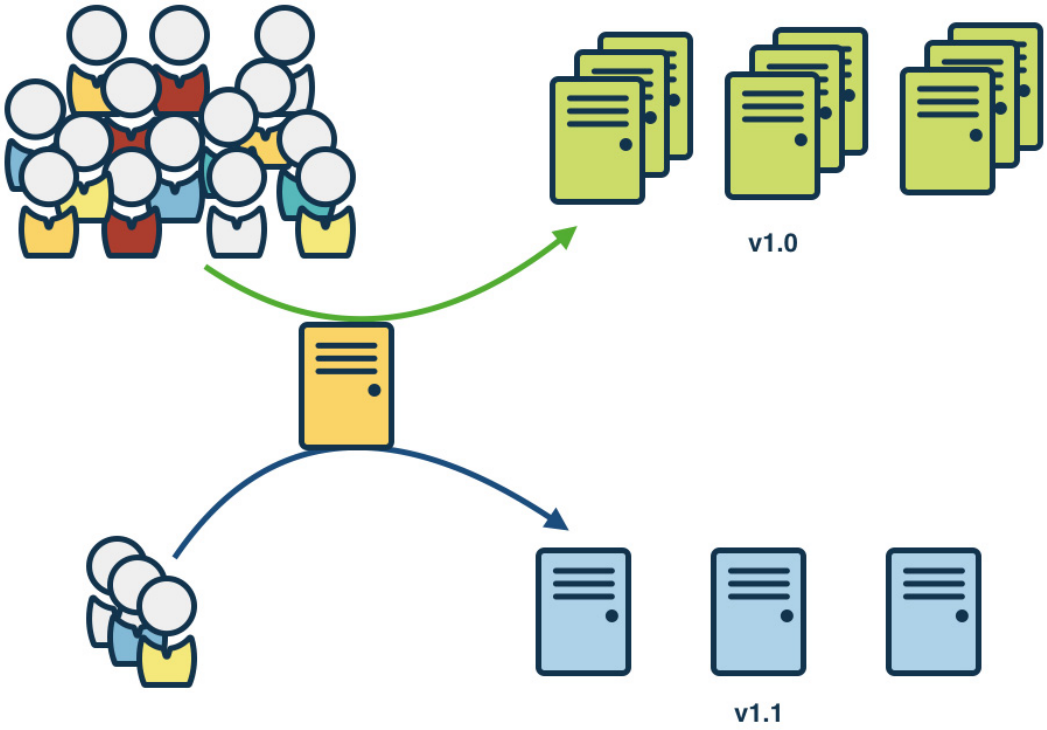

[MI-2609] IOException handling #6562
Edit
Open
pankajsinghal wants to merge 1 commit into master from cleaning
Conversation 0
Commits 1
Checks 0
Files changed 16
+1 −19
pankajsinghal commented 16 days ago
No description provided.
removed unnecessary IOException throw
331c3db
pankajsinghal requested review from ashishmjn and aditya-bsb 16 days ago
pankajsinghal changed the title from removed unnecessary IOException throw to [MI-2609] IOException handling 4 hours ago
pankajsinghal assigned aditya-bsb 4 hours ago
pankajsinghal added the TestRequest label 4 hours ago
aditya-bsb approved these changes 4 hours ago
View changes
hike-gitbot added TestPass and removed TestRequest labels 2 hours ago
Add more commits by pushing to the cleaning branch on hike/hike-mq.
Changes approved
1 approving review by reviewers with write access. Learn more.
Show all reviewers
All Checks have passed
2 successful checks
Show all checks
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Reviewers
aditya-bsb
ashishmjn
Assignees
aditya-bsb
Labels
TestPass
Projects
None yet
Milestone
No milestone
Notifications
Unsubscribe
You're receiving notifications because you authored the thread.
3 participants
Lock conversation

PR Passed

4. Taking code to production

Last but not the least, is how you take code to production system.

- Do a Canary deployment. Route small traffic to the new code base and slowly increase traffic to 100%.



- Monitoring and Alerting

  ➡ For monitoring, we have build in-house tools over Grafana and BigQuery for tracking both tech and product issues. Also, we use Overops for exception detection.

  ➡ For alerting we are using Nagios and VictorOps.

In next blog of “Availability series”, we will deep dive into specific components of our messaging architecture and how we improved on availability.