ICA0002: IT Infrastructure Services

High availability

Roman Kuchin
Juri Hudolejev
2025

Availability

Probability that the system works as required

Different ways to count:

- May include downtimes for a planned maintenance, or
- May exclude them

Part of system reliability engineering

Mentioned in the service SLA

- Example SLAs: AWS, Atlassian, Google Workspace, Pipedrive, Slack

Availability

Let's say that some system may be unavailable less than 1% of the time

System is available at least 99% of the time

In the observed period it makes less than

- 36 seconds per hour (in average)
- 14 minutes 24 seconds per day
- 1 hour 41 minutes per week and so on

Availability

Let's say that some system may be unavailable less than 1% of the time

System is available at least 99% of the time -- **"2 nines"**

In the observed period it makes less than

- 36 seconds per hour (in average)
- 14 minutes 24 seconds per day
- 1 hour 41 minutes per week and so on

"Nines"

Nines	Availability	Allowed downtime			
		Daily	Weekly	Monthly	Yearly
1	90%	<2.5 hours	<17 hours	~3 days	<40 days
2	99%	<15 min	<2 hours	<8 hours	<4 days
3	99.9%	<2 min	~10 min	<45 min	<9 hours
4	99.99%	<10 sec	~1 min	<5 min	<1 hour
5	99.999%	<1 sec	<10 sec	<30 sec	<6 min
...	...	...	...	...	...

Availability

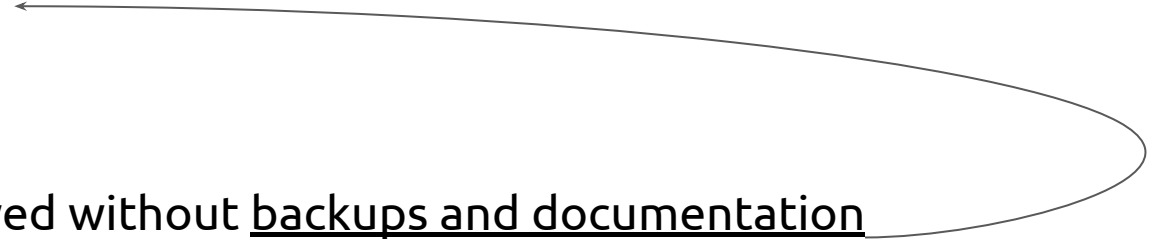
How to get more "nines":

- Less unexpected downtimes
- Faster problem detection
- Faster recovery

Availability

How to get more "nines":

- Less unexpected downtimes
- Faster problem detection
- Faster recovery



90% cannot be achieved without backups and documentation

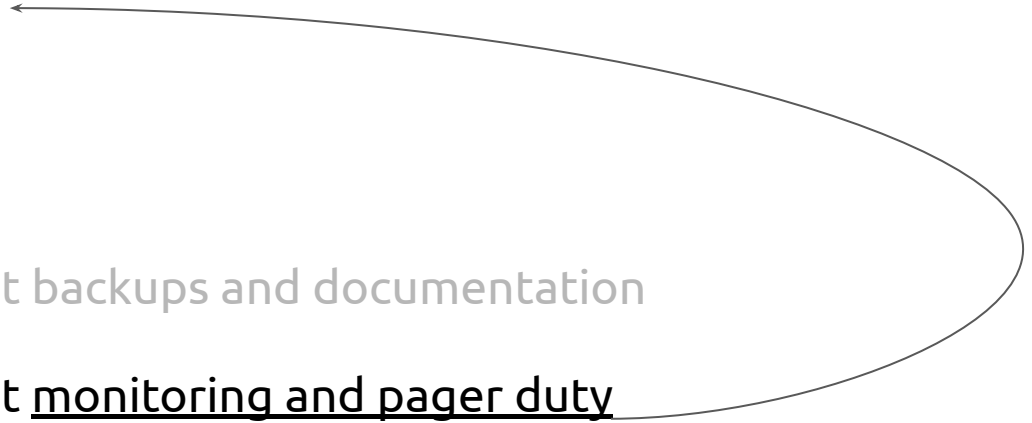
Availability

How to get more "nines":

- Less unexpected downtimes
- Faster problem detection
- Faster recovery

90% cannot be achieved without backups and documentation

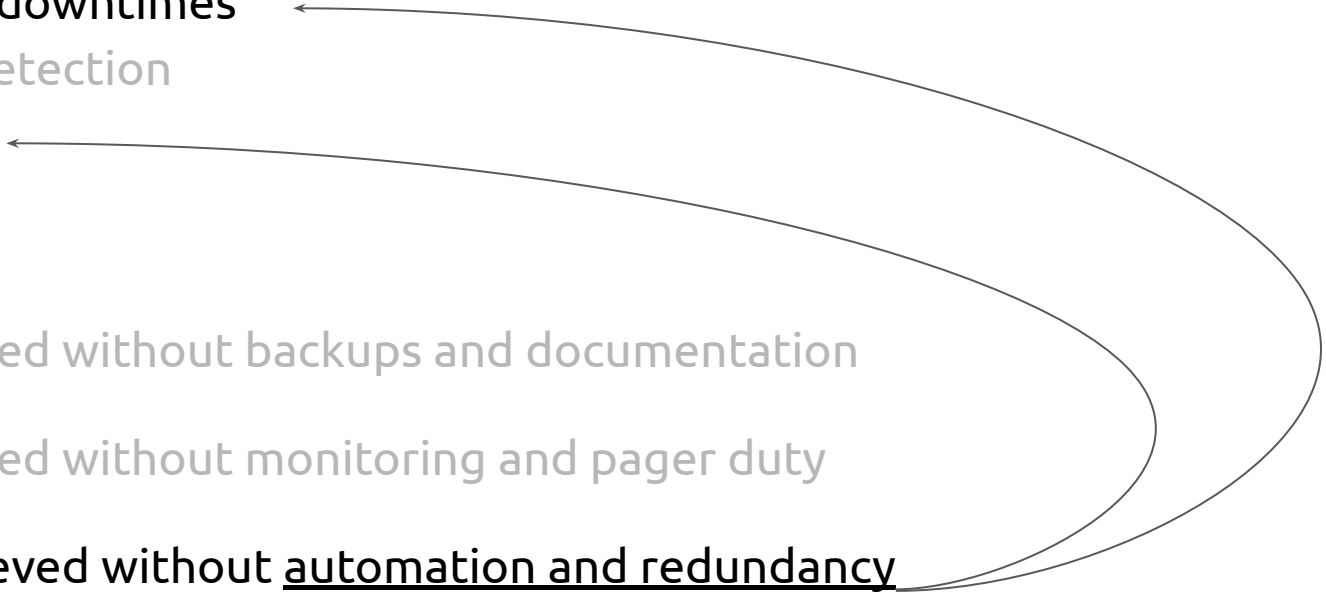
99% cannot be achieved without monitoring and pager duty



Availability

How to get more "nines":

- Less unexpected downtimes
- Faster problem detection
- Faster recovery



90% cannot be achieved without backups and documentation

99% cannot be achieved without monitoring and pager duty

99.9% cannot be achieved without automation and redundancy

Availability

How to get more "nines":

- Less unexpected downtimes
- Faster problem detection
- Faster recovery

90% cannot be achieved without backups ✓ and documentation ✓

99% cannot be achieved without monitoring ✓ and pager duty

99.9% cannot be achieved without automation ✓ and redundancy

Redundancy

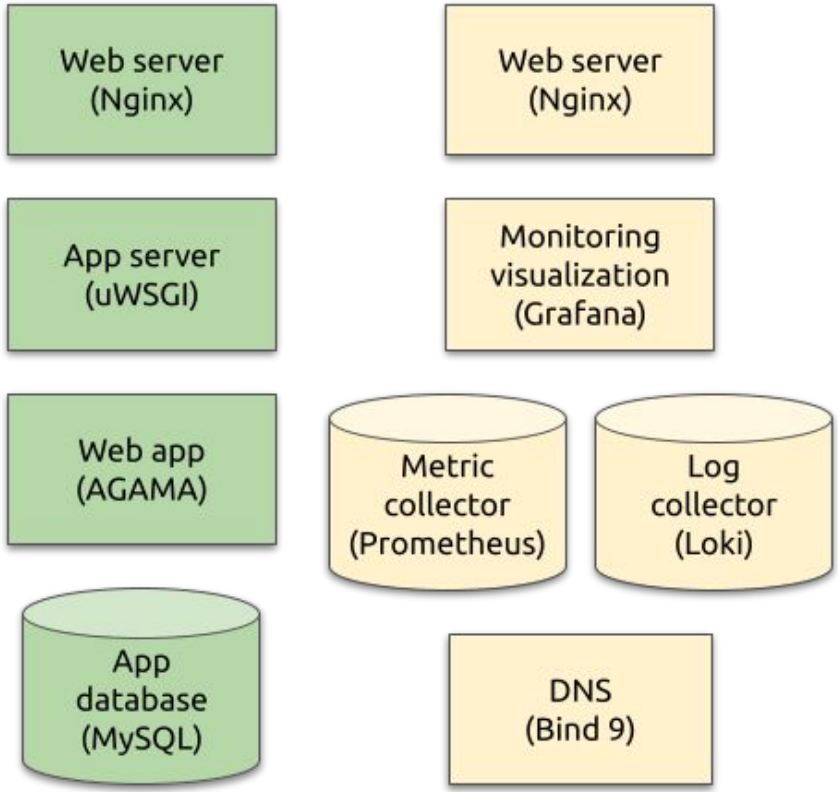
Duplication of critical components of the system

N+1

2N

2N+1

Current setup



MySQL redundancy

Replication (N+1, 2N)

- 1 primary node (write enabled; also called 'source', 'master')
- 1 or more read only replica nodes (also called 'slaves')

Clustering (2N+1)

- 3 or more primary nodes (all write enabled)
- May also have read only replicas

MySQL replication

Available on any modern MySQL flavor (Oracle, Percona, MariaDB) out of the box

Load balancing: separate instances(s) for intensive reads, reports, backups etc.

- Application must support it!

Manual failover in case of primary node failure

- Can be automated but needs additional work

Single write enabled primary node -- single point of failure

More info: https://dev.mysql.com/doc/refman/8.0/en/replication.html

MySQL asynchronous replication

Client sends query to MySQL primary node

MySQL primary node:

- Accepts and executes the transaction
- Writes the transaction to the binary log if this transaction changes the data (INSERT, UPDATE, DELETE queries)

MySQL replica:

- Connects to the primary node to get the latest changes from binary log
- Updates local copy of the data with the changes from the primary node

MySQL asynchronous replication

Faster query execution:

- Synchronization is done in a separate thread, does not block transactions

Network delays between members do not slow down queries:

- Allows geographically distributed replication

Data integrity is not always guaranteed

- Data loss occurs if primary fails before the replica has pulled the updates

MySQL semisynchronous replication

Client sends query to MySQL primary node

MySQL primary node:

- Accepts the transaction
- Notifies replicas if this transaction changes the data
- Waits until at least one replica confirms it has accepted the change
- Commits the transaction and sends the result to the client

More info: https://dev.mysql.com/doc/refman/8.0/en/replication-semisync.html

MySQL semisynchronous replication

Better data integrity guarantees

Longer query runs:

- Primary waits for data replication before commiting the transaction
- Query execution time depends on network delays (RTT)
- Query execution time depends on the fastest replica performance

Demo!

MySQL replication challenges

Your application must support it

- Different endpoints for different SQL queries
- Microservice architecture

Failover detection and automation

- Home-made scripts
- Common tools: Corosync + Pacemaker, Consul/Etcd, DNS
- Specialized tools: Orchestrator
- SQL proxies: HAProxy, ProxySQL etc.

GitHub case: https://github.blog/2012-09-14-github-availability-this-week

MySQL clustering

Several primary write enabled nodes: 3, 5, 7 etc.

- Should be odd number to avoid split-brain

Synchronous data replication

Widely used applications:

- MySQL NDB
- MySQL Galera Cluster / Percona XtraDB Cluster / MariaDB Galera Cluster

MySQL clustering challenges

Performance: cluster is as slow as its slowest member

Dependency on network delays

- Geographically distributed clusters can be a problem

Simultaneous distributed write queries: hotspots and deadlock errors

Database engine and schema restrictions

- InnoDB and XtraDB only
- Known LOCK and DELETE query limitations

MySQL redundancy

Replication

- Cheaper way to balance the load and simplify the **recovery** after the failure

Clustering

- More expensive and complex way to **prevent** the system downtimes

Can be used together in complex systems with high availability requirements

More reading

How MySQL replication works, in details:
https://dev.mysql.com/doc/refman/8.0/en/replication-implementation.html
and subpages

Galera cluster pros and cons:
https://www.percona.com/blog/2013/05/14/is-synchronous-replication-right-for-your-app

So how many "nines"
is high availability?

High availability

"High" in HA does not mean a certain number of nines

High availability is a characteristic of a system, and a principle of system design

- Better uptime
- Tolerate more failures
- Faster recovery

Don't create complex systems.

Create simple systems that scale well.

HA does not replace backups!

Questions?