severalnines

The DevOps Guide to Database Management

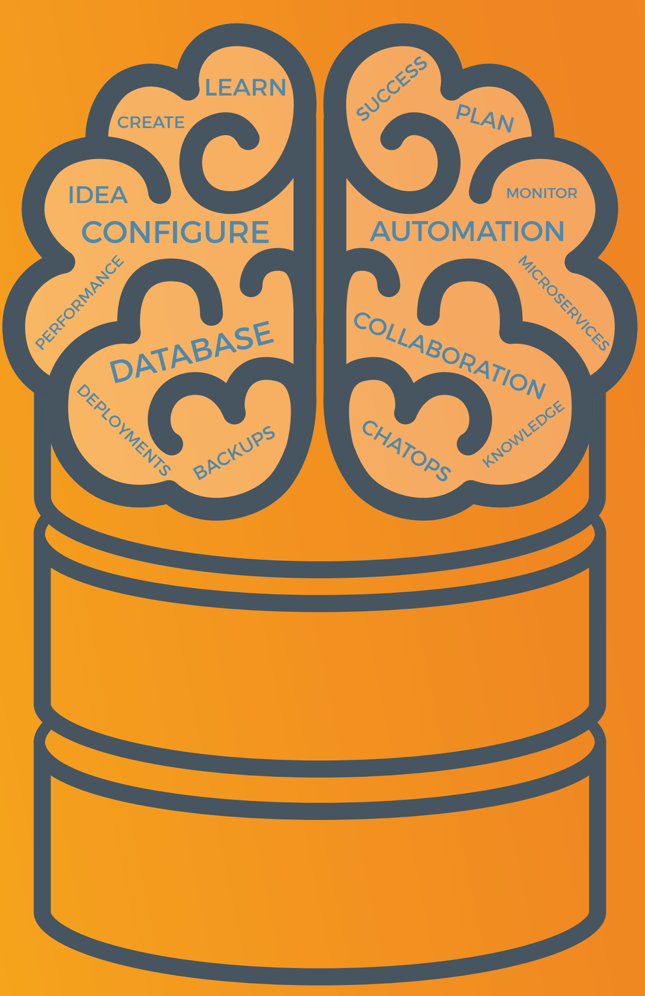


Table of Contents

1. Database challenges with Devops 4
    1.1. Database collaboration in DevOps 5
    1.2. What does this mean for databases? 5

2. The impact of microservice architectures 6

3. Managing databases in a DevOps environment 8
    3.1. Deployment automation 8
    3.2. Performance monitoring 9
    3.3. Schema changes 9
    3.4. Version upgrades 10
    3.5. Automated Failover 11
    3.6. Data distribution 11
    3.7. Managing data flows 13

4. ChatOps 14

5. Summary 15

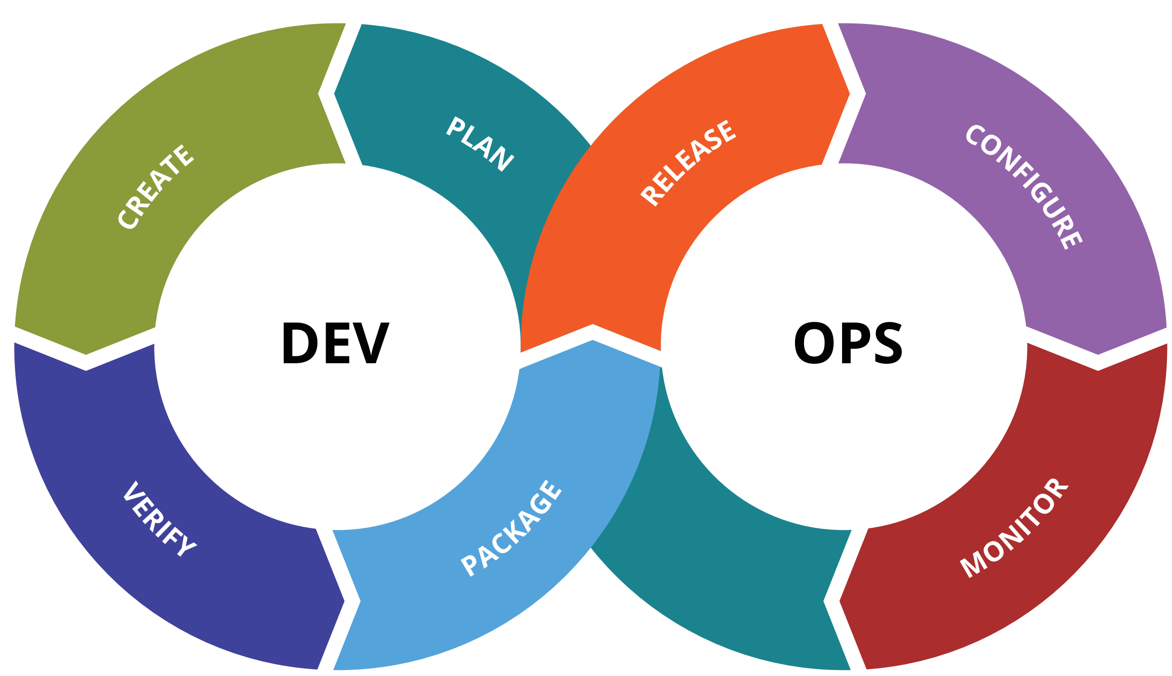
6. About ClusterControl 16

7. About Severalnines 16

8. Related Resources from Severalnines 17

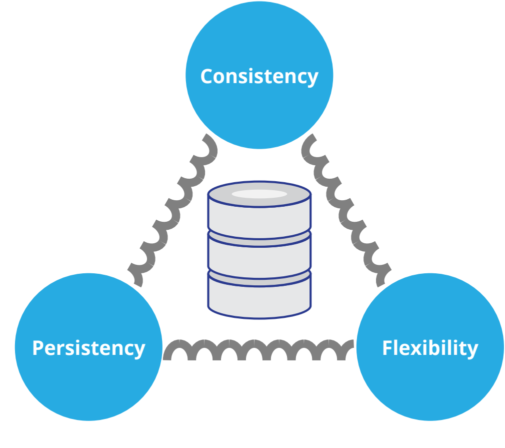
Database challenges with Devops

DevOps has become one of the fastest growing terms in IT for the past five years, and this comes to no surprise. DevOps is a term that is a compound of the words development and operations, where it refers to the collaboration and communication between (software) developers and information technology operations professionals. It changes the way these two groups of people work in a cultural and environmental way. This improves the building, testing and releasing of software and allows more reliable, frequent and rapid deployments.



In devops the traditional development process is changed to a continuous process of developing and releasing the product. This circular process is much more suitable for short development cycles, where the cycle iterates over code (plan), build (create), test (verify), package, release, configure and monitor.

As quick and continuous release cycles dictate frequent updates from development to production, how do databases fit in this picture? Most RDBMS databases are mostly built around securing the integrity of the data. This means that certain trade-offs have been made on how the database copes with (schema) changes. In general any change to the structure of the data in a RDBMS will involve locking and take a painfully long to apply. To overcome this problem, many DevOps will rather favor schemaless datastores like MongoDB. Schemaless datastores have made the trade-off of flexibility over consistency.



1.1. Database collaboration in DevOps

As the DevOps method requires more close collaboration, this shifts a lot in the roles of the people that are part of the team and some of them will now share parts of their roles. The developer not only develops the application, but also is part of the release process. The developer is more likely to know most about the application, so the developer should also know how to monitor the application best. Similarly, QA has to know what they are going to test, how it works and what environment it is being hosted in. The operations members will likely know more about the internals of the application they are hosting, saving valuable time when troubleshooting issues.

With the shift in roles, additional responsibilities will be inevitable. The developer can no longer hide behind the fact that "she is just the developer" when it comes to operational issues, similarly for the other roles. As the whole team now owns the application, the team members will also feel more connected to the product that they are responsible for. Naturally they will feel more responsible when an outage occurs.

So who will be responsible for the role of DBA in a DevOps environment? In most cases this will be a collaborative role between the developer and the ops roles. The developer will drive the initiative, changes and performance aspects while the ops will handle the consistency and security.

Who is the DBA in a DevOps environment?

1.2. What does this mean for databases?

As mentioned before relational databases are less flexible by nature, while DevOps actually requires more flexibility. It will be a continuous trade-off between Dev and Ops for the solution chosen. Regardless of the solution chosen, there are many challenges that need to be overcome.

The first challenge will be deployment automation. As continuous deployments will be part of the DevOps process, the entire environment needs to be fully automated, provisioned and primed with the necessary data.

The second challenge will be the incompatibility between relational databases and microservice architectures. Microservices have, by definition, a shared-nothing architecture. The reasoning behind this is to lift any dependencies on other microservices and to prevent the outage of one microservice to affect the other. This means that in its purest form a microservice will be nothing more than a single table in a single schema on a database cluster.

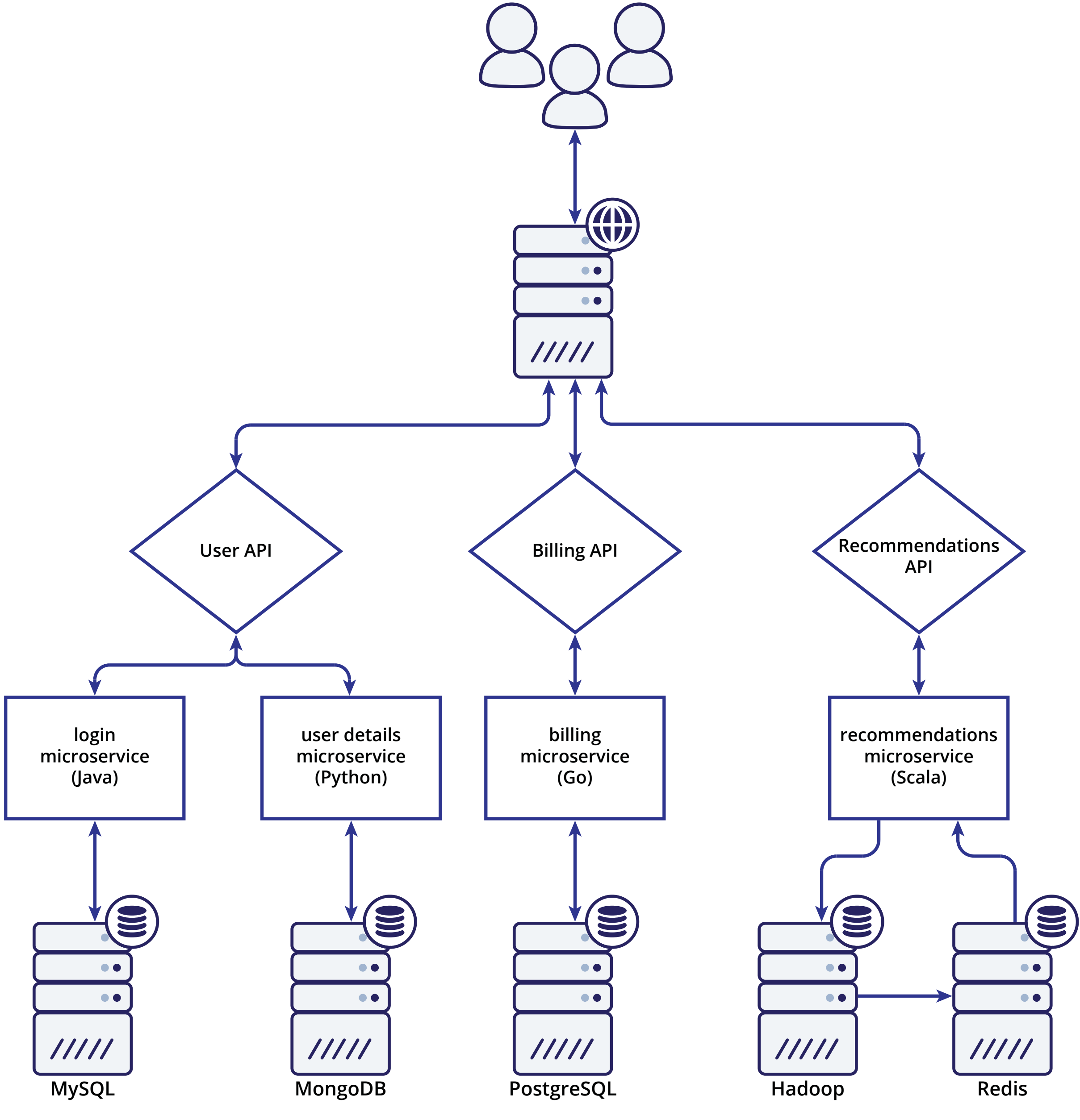
The third challenge is collaboration. Collaboration between the members of a DevOps team will be key to its success. Communication is the most important element in collaboration, so it is essential that every member of the team is up to date with the latest information. Communication channels, also known as ChatOps, will play a big role in this.

In this whitepaper we will touch upon each of these challenges one by one. We will also see how Severalnines ClusterControl can be used to address these challenges.

The impact of microservice architectures

Microservices are embraced fully by DevOps as they are easy to deploy, have a strong cohesion and loose coupling. In their philosophy they share nothing, have small and simple schema designs and are interchangeable with other data stores. This means their features are very generic, not storage specific and this makes them easy to deploy.

Even though microservices shouldn't share data between each other, this poses an architectural challenge. How would you get data that is stored in another microservice? If one microservice depends on a piece of data that is stored in another service, the simplest solution would be to query the data directly from the database of the other service. From a development perspective this is the easiest and quickest solution, but



it actually poses a big problem. It would create a dependency from one microservice to another. Any change to the data structure or infrastructure of the depending microservice could create a direct issue.

For example, when a DevOps team, responsible for microservice A, decides they need to scale the database, move data to another datacenter, failover the master to a slave or even change the schema to satisfy new requirements, the depending microservice B should be aware of any of these changes. This probably results in microservice B failing to retrieve the correct data and results in throwing an error. As you can see, the data layer within microservices is dynamic, so you do not want any consumers of data of these services to be depending on their internal structure. Instead these microservices are supposed to retrieve data only via the API of the microservice.

This abstraction introduces another challenge to their architecture: if they are depending on the API from another microservice, they might be waiting for a request made to an API of another dependent service. If the dependent microservice responds slowly, or throws out errors due to their database layer, this get propagated upstream as well. This means these microservices need to be made resilient against any API errors that may occur from the start. The added benefit is that this abstracts the entire data layer away from the frontend.

The benefit of this, is that now the microservices can be independently changed, sharded or scaled and data locality is no longer an issue. This also opens up the ability to utilize specific features of a non-standard datastore when necessary. If it is necessary to solve a scalability issue by moving from MySQL to MongoDB, this would not pose a problem anymore for all depending microservices.

ClusterControl supports various different database topologies that would satisfy most storage needs. Bringing up a new database for a microservice can be done in minutes and scaling of the database has been built in from the start. ClusterControl is also multi-datacenter ready, so scaling beyond a single datacenter is no longer an issue.

Managing databases in a DevOps environment

As DevOps requires a whole different approach of working, the focus lies mainly on automation. As developers and ops need to bring-up and tear-down environments frequently, this means that bringing up an environment needs to be a (fully) automated process. If a new build environment is necessary, it should be deployed within minutes. For the database environment this might pose a problem, as the databases are mostly used for persistency and consistency of data.

3.1. Deployment automation

Installing database software can be tedious. The initial software installation can be a breeze, but after this step a lot of work awaits. Customization, configuration, tuning and setting up (replication) topologies are only a few of these examples. Apart from the fact that a manual setup takes a lot of time, it also is error prone. After everything eventually has been setup, maintenance becomes another hurdle. Automation of systems is therefore key, and this starts with deployment automation.

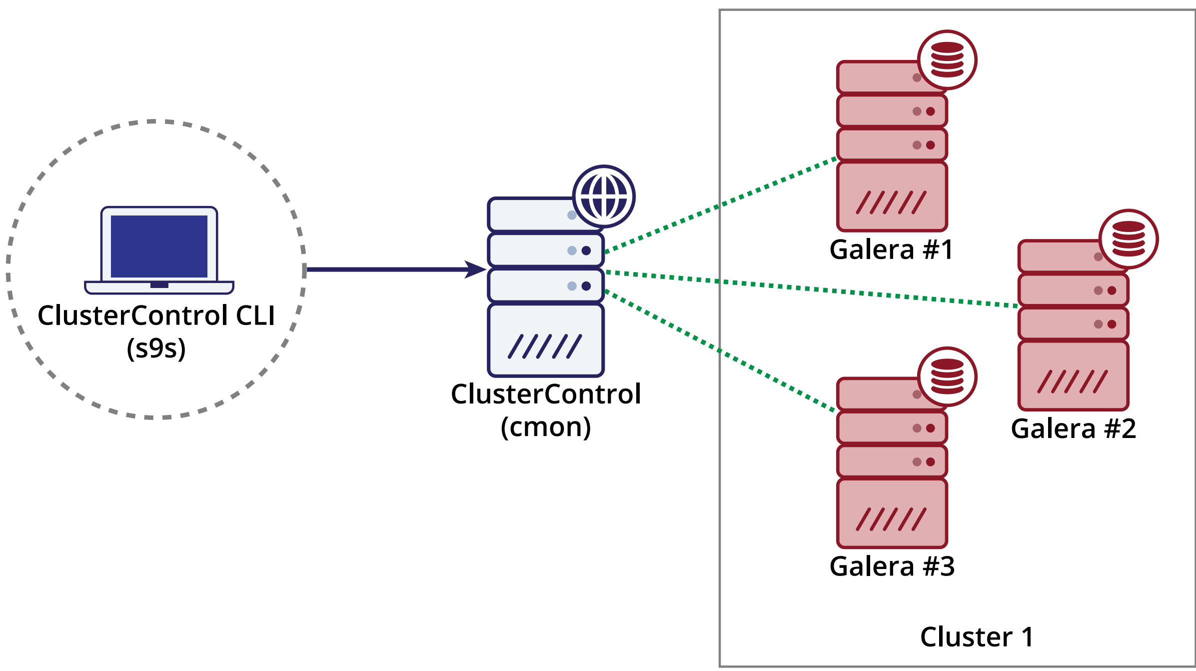
Deployment automation can easily be achieved by frameworks like Puppet, Chef and Ansible. These frameworks are excellent for deploying packages and static (config) files. However this does pose a problem with databases. Database packages can be installed, configuration files deployed and managed, however databases also require meta information like users, ACLs and password management. Also in more complex environments involving clustered database nodes and high availability components like load balancers, it requires a more holistic approach.

Bringing up an environment doesn't just end with the deployment of software. Different environments, like development, QA and Disaster Recovery, have different purposes. If an environment needs to be fully deployed, it also includes additional settings and data to be populated. Data and settings can be extracted from existing backups, but it will be quite difficult automating the extracting and filtering of records from backups. It will not be an easy task to automate this inside a configuration management tool like Puppet or Chef, as those frameworks are mainly built for deployment only. This type of automation would be possible with Ansible, as Ansible has more scripting, dependencies and workflow options. However these scripts need to be maintained, reviewed and altered for every change performed in the database topology or in the deployment of the application. It would be much easier if you could simply clone an existing environment into a new one and then remove or obfuscate data after cloning.

ClusterControl is able to automate the deployment of a clustered database environment, as well as clone an existing database cluster onto a new set of nodes, without downtime of the source cluster. This makes the deployment of on demand environments much easier.

With the ClusterControl command line interface (CLI), the user can send commands to the ClusterControl backend and retrieve the current status. This makes the automation

part of bringing up entire environments even easier. Now anyone can script the creation of the full stack with only a few lines of code. Especially with environments that consist of many loosely coupled components, like microservices, this can save a lot of time.



3.2. Performance monitoring

Once you have the overview of the full stack of the application and services, you can truly monitor everything that is happening in your system. Especially with Microservices this could easily be separated on a per-service dashboard. The benefit of this separation would be that any problems can easily be isolated or pinpointed to a single service. Once identified the impediment could be lifted and performance restored back to normal.

In general developers have a preference for application performance monitoring systems like New Relic and appDynamics. These products do give the entire full stack overview, but do they actually provide the necessary database insights? These products only measure from the application point of view, but have no ability to drill down further into the database metrics. For this you would still depend on a specific database monitoring system. This would then get into true DevOps: developers and ops resolving performance issues collaboratively.

3.3. Schema changes

As mentioned in the previous section, schema changes are inevitable within DevOps. Whenever a new version of the application or microservice requires an additional field to be stored, the change is foreseeable and simple to perform. It really poses a problem once fields are changed or removed. This means the structure of the table is changed and fields may not be returned in the expected manner.

Especially when an ORM is used, this will require frequent schema changes whenever an object is altered. Most ORMs will auto-generate schema changes and queries based upon their internal code. In some cases these ORMs will only function properly when given grants to alter and modify schemas, otherwise they will produce a fatal exception

that the schema they are expecting is not according to the one found in the database. The simplest solution would be to grant these ORMs schema modification rights, but this makes it complicated to stop the ORM from performing schema changes on the fly.

Schema changes would create internal locks in the database server, resulting in other queries being queued and piling up until the database server runs out of resources. And even if the schema changes are properly applied via online schema change applications, like pt-osc and gh-ost, it could still create performance issues due to the difference in schema.

Therefore detection of unauthorized schema changes is essential. Monitoring the layout of your schemas isn't that difficult, but detecting the change is a bit harder. A schema change detection feature is available in ClusterControl, so you will be alterted if a schema gets altered.

Detection is the same as prevention, and often the impact of a schema change is only visible once it gets deployed to production. As a pre-emptive measurement you can load test your schema change by mimicking your production workload using a benchmark tool like sysbench. If you are already using a proxy solution, in some cases you could copy all queries to a second database node or cluster to see the impact of your schema change. In the case of ProxySQL it is possible to even mirror the entire workload to another server and do complete workload analytics, based upon the query time.

3.4. Version upgrades

Version upgrades are very important for keeping the database software up to date. In the new version, bugs and vulnerabilities may be resolved, stability improved and performance may be far greater than the current production version. These all are valid reasons to upgrade the software to the latest version.

While regular version upgrades are routine for DBAs and sysops, in DevOps this may actually pose a similar challenge as with schema changes: who is responsible for these and how do we apply them safely to a running production system? Like schema changes, the solution for this isn't going to be an easy and clear one. Whist having to retain high availability, the system may be vulnerable at the same time.

You won't be dealing only with major version upgrades, though - it's more likely that you'll be upgrading to minor versions more often, like 5.6.x -> 5.6.y. Most likely, it is so that the newest version brings some fixes for bugs that affect your workload, but it can be any other reason. There is a significant difference in the way you perform a major and a minor version upgrade.

A minor upgrade is relatively easy to perform - most of the time, all you need to do is to just install the new version using the package manager of your distribution. These upgrades are easy to script and coordinate. ClusterControl contains version upgrade functionality that supports these minor updates.

For a major upgrade you may be required to perform various tasks and/or conversions after you have upgraded. For instance with MySQL it is advised to dump all contents of your database and load it into the new version after you have upgraded. Naturally major upgrades will be unique per use case, and therefore not suitable to be

automated. However if you have to perform the major upgrade many times on many separate clusters, it would be beneficial to create a document with the outlines of the upgrade process and automate this as much as possible

3.5. Automated Failover

Making a microservice resilient, means that the loss of a database node should not interfere with the service. A small outage, like a failing slave node, should hardly be noticeable from the application point of view. If there is a major failure in the topology, like a master node crashing, election of a new master and failover to another node needs to happen within seconds. Some users might experience a glitch, but the application will remain stable. There are various ways of making the microservice resilient against database failures, and most of these reside in the way of connecting to the database nodes.

If the microservice is cluster aware, the application would need to be configured with all available nodes and it would be required to keep state of each node in the cluster. This will put an extra strain on the application, as it now needs to understand the underlying database software and its ways of failing over. It all depends on the method of implementation and it might be error prone.

To overcome this, the application could be configured using a resource manager like Zookeeper or Etcd. This resource manager will then keep track of the state of the cluster and would always provide the correct configuration to the application. Even though this is a very popular method used in script languages that lack their own internal state, for languages like Java who keep state this might pose a problem. A database ORM layer, like Hibernate, might not be able to cope with sudden changes in roles of open (persistent) connections. A node that once used to be a master has now become a slave, so how would the ORM cope with such a change? To get around this, this might require additional logic inside the application that would solve issues like this.

It would be far better to have a transparent and invisible process that manages the availability of the cluster. This is where a load balancer or database proxy would come in handy. The load balancer and proxy will closely monitor all nodes in the cluster and retain availability for the application by redirecting connections to the appropriate nodes in the cluster. The failover itself will be managed by another external process, to ensure that the proxy component isn’t a single point of failure. ClusterControl can facilitate the automated recovery and failover in replicated or clustered setups, while it also is able to deploy the load balancers and proxies with the appropriate healthchecks.

3.6. Data distribution

We mentioned data locality in a previous paragraph. Data distribution plays a big role in DevOps. As the environments and schemas may seem to be getting more simplified by the Microservices architecture, the complexity of data distribution increases with the introduction of multi-datacenter, functional sharding and horizontal sharding.

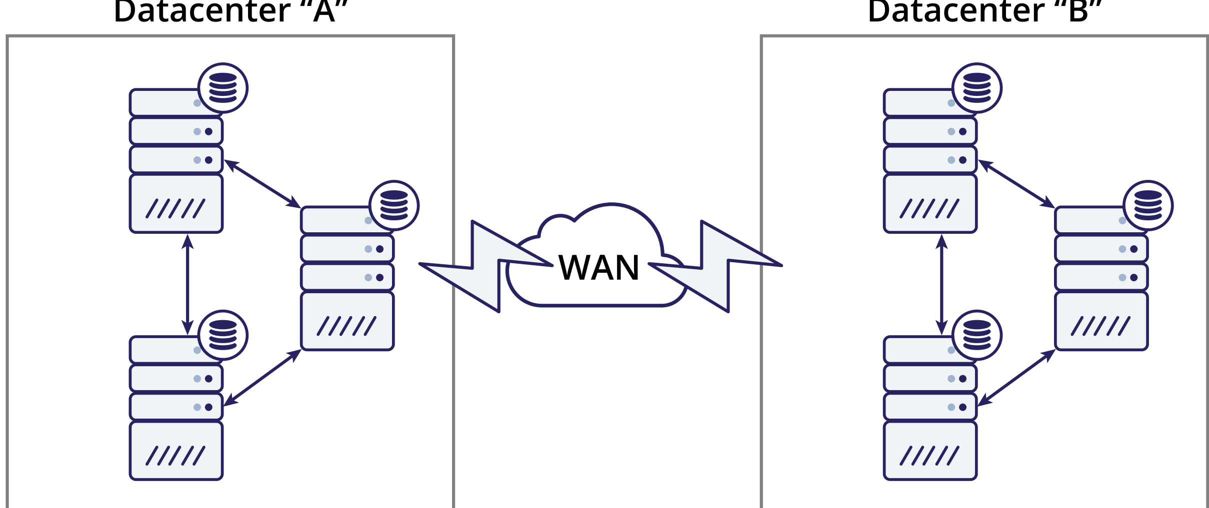
First of all the microservices dictate the functional sharding of data. Every microservice is supposed to run contained in their own schema and tables. As your end users will utilize multiple microservices, their data will be distributed amongst the various schemas. Unless all your microservices use the exact same database cluster for storage, this functionally shards your data between the various datastores used.

Some microservices may show uneven usage. While some of them only contain the user data, others may store details or even log-/click-data on every interaction with the user. This means every microservice may show different query- and growth patterns. Once a microservice becomes larger than the storage capacity of the used database cluster, a new scaling approach needs to be taken. If additional future growth is expected, horizontal sharding has to be considered. This means the data of the microservice is no longer contained within a single database cluster, but rather spread over a group of clusters.

Application sercer
read/write
read/write
read
Master
Slave
Original replication setup
read/write order_status_log
Master
read order_status_log
Slave
Second replication setup

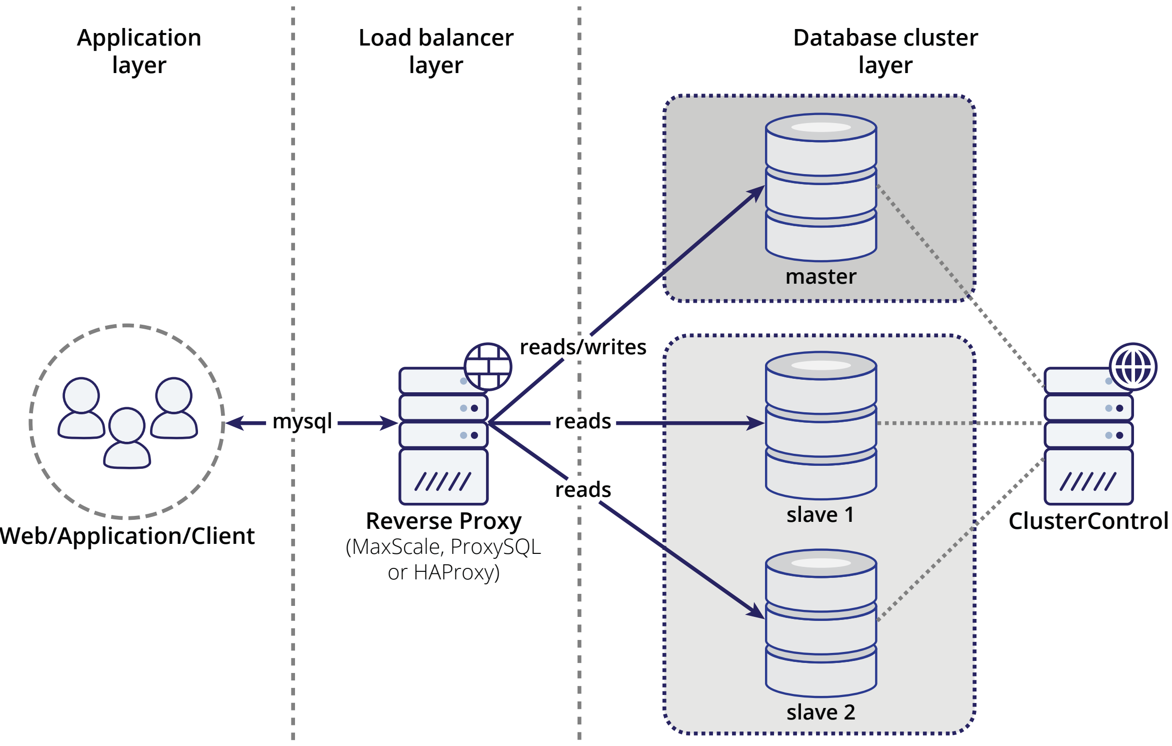
Whether infrastructure grows over time or availability is demanded, multi-datacenter is something to keep in mind. If data is stored over multiple data centers, this means the data locality may not be present. Moving data from one site to another might pose a risk in both availability and security. Make sure your multi-datacenter strategy includes failover scenarios and secured connections (VPN, SSL).

In a previous paragraph we spoke about schema changes, and once your databases are sharded schema changes will become a lot harder to perform. As the application or microservice reads and writes from multiple shards, it now becomes essential to orchestrate your schema changes in a proper way. It almost become unavoidable to make the application or microservice schema aware, where it understands the various versions of the schema used to prevent any fatal exceptions once it does not comply to this.

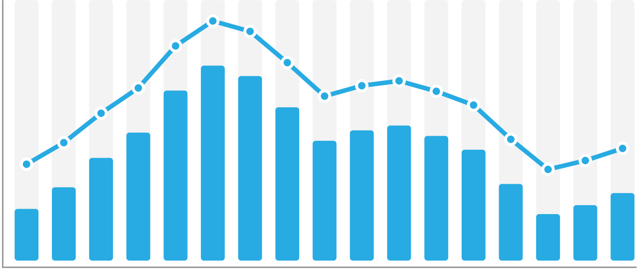


3.7. Managing data flows

Microservices abstract complexity away from the application using them, but the flip side is that the complexity of the infrastructure increases at the same rate. We mentioned data locality, sharding and schema change so far, and these are just a few examples of how the complexity can increase. To make the underlying infrastructure more transparent and easy to use, is the use of a load balancer or proxy that we have described earlier. With the help of these you can manage the data flows more easily and isolate the database infrastructure from the application.



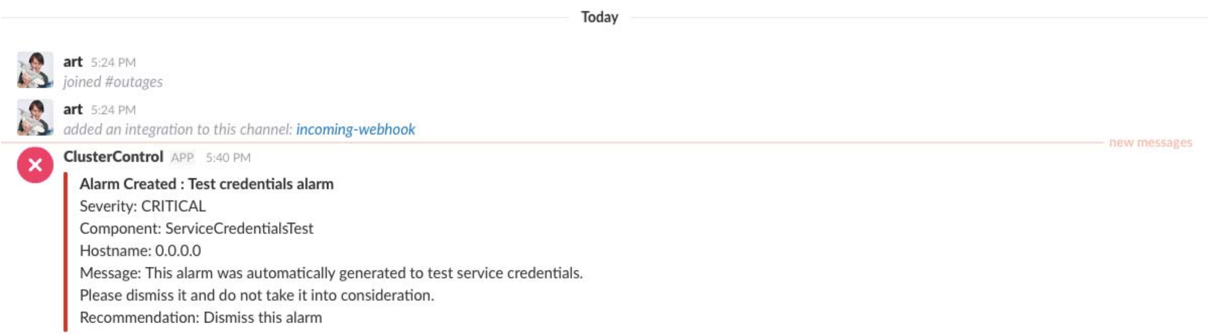
Using a proxy can also help you perform maintenance on your infrastructure. If you are working on one node, another can take over its role in the cluster. The proxy will detect the node you are working on is no longer available, and reroute the traffic to another available node.



Proxies can also help with scaling. Once you add a new node to a cluster, it will automatically receive traffic from the proxy. This way the application doesn't need to be aware of the nodes it can reach, and the scaling of the cluster becomes an automated process. For the application it doesn't matter whether it is sending its queries to 1 or 20 servers: as long as the proxy responds within reasonable time it will be fine. This brings us to another benefit of using a proxy for these cases: autoscaling clusters becomes possible this way. With workload analysis on the proxy, you could auto-provision new nodes and add them to the cluster when workloads are becoming too big to handle for the cluster. Once the workload is low enough, you could remove and de-provision these nodes again.

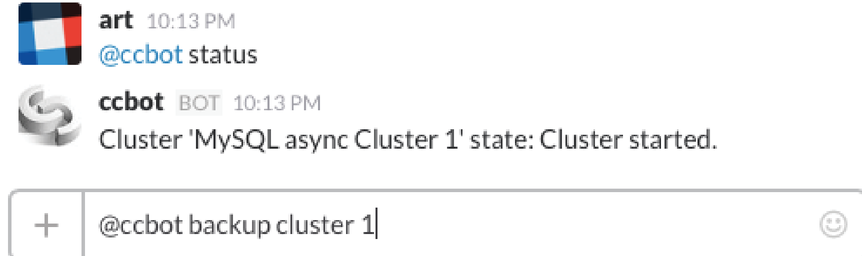
ChatOps

One of the most important aspects of DevOps is the communication between team members and other teams. To improve communication, often many chat channels for the various teams and microservices are used. If anyone deploys a new version, it is customary to communicate and coordinate the deployment via the chat as well. Anyone responsible or involved in this process will be put in the same channel, that if anything goes wrong during deployment the appropriate person is immediately available.



Since databases are an integral part of applications and microservices, it will be a large part of the discussions in the channel. It would improve the situation if the database would also take part in the channel, and you can do this by providing as much information about the database to the discussion. Status updates, database health, backup status and high availability are just examples of the information you could let contribute to the discussion. ClusterControl can easily integrate in the most used chat platforms via the ClusterControl Integrations.

Since ChatOps is bi-directional, it would also improve if users of the channel were able to send messages to the database as well. For example, if the team creates a new backup prior to deploying a new version of their application, a snapshot of the previous state would be available if a rollback is necessary. For this the ClusterControl commandline can easily be integrated using a chatbot. The ClusterControl CCBot modules can either be installed on your existing chatbot, or installed as a standalone chatbot using the Hubot framework.



Summary

Even though in more traditional environments developers and DBAs work on the opposite part of the chain, in DevOps they actually have to collaborate. This may feel unnatural at first, but it actually makes sense once you start to perform true DevOps: separation of function through microservices. The isolation of (performance) issues through microservice will make it much easier to resolve and overcome them. Frequent deployments ensure these problems are resolved much faster than in a traditional development environment. Even if these frequent deployments require as often schema changes, applying them through automation will make it much easier to cope with them. And as long as all members of the same DevOps team are collectively working through ChatOps, nobody will miss an update.

About ClusterControl

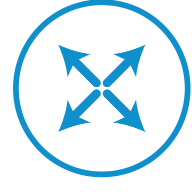
ClusterControl is the all-inclusive open source database management system for users with mixed environments that removes the need for multiple management tools. ClusterControl provides advanced deployment, management, monitoring, and scaling functionality to get your MySQL, MongoDB, and PostgreSQL databases up-and-running using proven methodologies that you can depend on to work. At the core of ClusterControl is it's automation functionality that let's you automate many of the database tasks you have to perform regularly like deploying new databases, adding and scaling new nodes, running backups and upgrades, and more.

About Severalnines

Severalnines provides automation and management software for database clusters. We help companies deploy their databases in any environment, and manage all operational aspects to achieve high-scale availability.

Severalnines' products are used by developers and administrators of all skills levels to provide the full 'deploy, manage, monitor, scale' database cycle, thus freeing them from the complexity and learning curves that are typically associated with highly available database clusters. Severalnines is often called the "anti-startup" as it is entirely self-funded by its founders. The company has enabled over 12,000 deployments to date via its popular product ClusterControl. Currently counting BT, Orange, Cisco, CNRS, Technicolor, AVG, Ping Identity and Paytrail as customers. Severalnines is a private company headquartered in Stockholm, Sweden with offices in Singapore, Japan and the United States. To see who is using Severalnines today visit:

https://www.severalnines.com/company



Deploy

Monitor

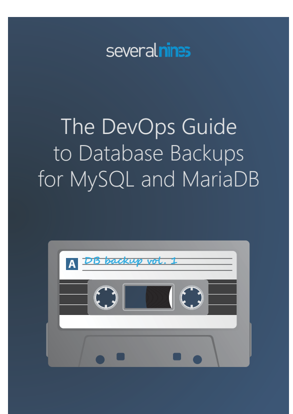
Scale

Related Resources from Severalnines

ClusterControl for SysAdmins & DevOps

ClusterControl helps SysAdmins & DevOps professionals with solutions to their database challenges. Learn about the ClusterControl features that address these challenges and what the benefits are that can be gained from using ClusterControl to automate and manage your open source databases.

Learn more



The DevOps Guide to Database Backups for MySQL and MariaDB

This whitepaper discusses the two most popular backup utilities available for MySQL and MariaDB, namely mysqldump and Percona XtraBackup. It further covers topics such as how database features like binary logging and replication can be leveraged in backup strategies. And it provides best practices that can be applied to high availability topologies in order to make database backups reliable, secure and consistent.

Download here

Webinar replay: 9 DevOps Tips for Going in Production with Galera Cluster for MySQL / MariaDB

Galera Cluster for MySQL / MariaDB is easy to deploy, but how does it behave under real workload, scale, and during long term operation? Proof of concepts and lab tests usually work great for Galera, until it's time to go into production. Throw in a live migration from an existing database setup and devops life just got a bit more interesting ...

If this scenario sounds familiar, then this webinar replay is for you!

Watch replay

severalnines

Deploy

Manage

Monitor

Scale

© 2017 Severalnines AB. All rights reserved. Severalnines and the Severalnines logo(s) are trademarks of Severalnines AB. Other names may be trademarks of their respective owners.