# DevOps for Database

BY SANJAY CHALLA

## INTRODUCTION

The way that software is developed, built, and delivered has gone through a profound transformation. Across industries, teams have moved to nimbler software engineering processes in an attempt to reduce time to market and improve cost effectiveness. To fuel this change, there has been an explosion in the growth of build and application release tools to enable DevOps processes with Continuous Integration and Continuous Delivery. In their 2017 survey, RightScale found that 84% of enterprises and 72% of SMBs are adopting DevOps, underscoring the ubiquity of the broader digital transformation movement across the marketplace.

The results speak for themselves: Organizations are making large investments in DevOps tools and processes to accelerate the delivery of capabilities to customers while maintaining quality and optimizing cost. Central to this transformation is the systemic identification and elimination of manual processes and tasks with automation. According to the 2017 State of DevOps Report, the firms that have been most successful at making the transition to DevOps have automated far more than their lower performing peers.

The high performers have automated 33% more of their configuration management, 27% more of their testing, 30% more of their deployments, and 27% more of their change approval process than their low-performing peers. As a result, the high performers have been able to deploy 46x more frequently and have 440x faster lead times, a 96x faster mean time to recovery, and a 5x lower change failure rate. By leveraging automation and DevOps practices, world-class enterprises are able to move much faster than their competitors while simultaneously delivering at a higher quality.

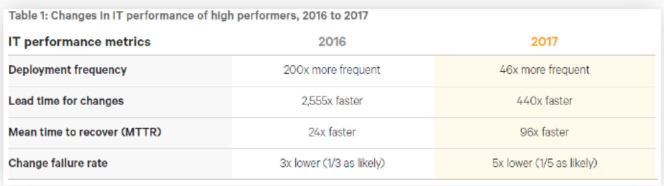


Table 1: Changes in IT performance of high performers, 2016 to 2017

| IT performance metrics | 2016 | 2017 |
|---|---|---|
| Deployment frequency | 200x more frequent | 46x more frequent |
| Lead time for changes | 2,555x faster | 440x faster |
| Mean time to recover (MTTR) | 24x faster | 96x faster |
| Change failure rate | 3x lower (1/3 as likely) | 5x lower (1/5 as likely) |

Directly from the State of DevOps report, this table quantifies the extent to which high performers outdo competition on key agility metrics that help their business outperform the competition in the market with quicker, more stable, and higher-quality software releases.

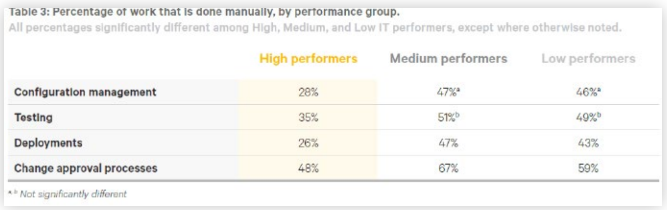


Table 3: Percentage of work that is done manually, by performance group.
All percentages significantly different among High, Medium, and Low IT performers, except where otherwise noted.

| | High performers | Medium performers | Low performers |
|---|---|---|---|
| Configuration management | 28% | 47%[a] | 46%[a] |
| Testing | 35% | 51%[b] | 49%[b] |
| Deployments | 26% | 47% | 43% |
| Change approval processes | 48% | 67% | 59% |

[a, b] Not significantly different

## THE DEVOPS DATABASE CHALLENGE

As tools, processes, and best practices permeate the marketplace, there still remain some notable barriers preventing many firms from realizing the true value of their hefty investments into DevOps. At the center of this issue is the myopic interpretation of what encompasses an "application." For many software teams, the data or persistence layer — more finely, the database in particular — has been effectively forgotten. As a result, while there has been a sharp focus and tremendous acceleration in the velocity of application software releases, updates to the underlying database have remained manual and are increasingly a bottleneck to the overall software delivery pipeline.

The omission of the database was not entirely accidental. Managing the database in a DevOps environment is very

Datical

# DevOps, Meet Database

Release Faster or Die.

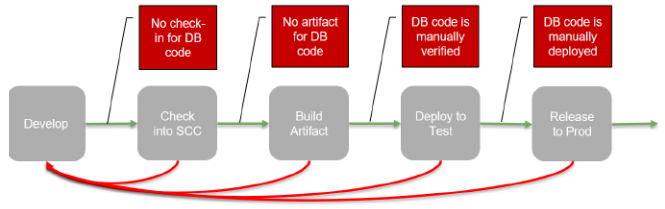
Database Release Automation

www.datical.com

challenging. While it's perfectly okay — and frankly espoused by Agile zealots — for developers to move quickly and "fail fast, fail often" with application updates, the same is not true of the database! Databases hold incredibly valuable state, and a bad database change can result in severe data loss, application outages/downtime, or security holes that can be exploited to exfiltrate sensitive business data. A careless change to a production database can be devastating both to a firm's finances and reputation. As an example, IDC reports that the average cost of a critical application failure per hour is $500,000 to $1 million, not to mention the damage to the brand reputation.

There's a reason that the last decade has seen an unprecedent growth in tools to enable rapid application software development while the database has remained effectively untouched by similar automation. The database has long been a black box, closely guarded by administrators chartered with the sole task of securing critical business data essential to the firm. Understandably, there has long been a strong aversion to change on the database side because change historically came with risk to the valuable business assets stored within the database.

With application software teams moving faster than ever, it's become untenable to treat the database like a black box surrounded with a heft of manual process. There are solutions that focus on the agility of the database which allow for rapid changes while reducing risk. The trick is to leverage the existing tooling and workflow already in place for application software and to add automation to the database. The goal is to treat database code just like application code instead of treating the database with a manual out-of-band process.



As application code transitions from development, through testing environments, and finally to production, the currently manual database change process can add significant delay, error, and risk to the overall software release.

## DEVOPS FOR THE DATABASE BEST PRACTICES: DATABASE RELEASE AUTOMATION

There exist a handful of commercial vendors and open-source projects aimed at bringing DevOps automation to the database. Whether you're opting to build a totally custom solution, extend an open-source project, or invest in a commercial off-the-shelf solution, the remainder of this guide focuses on key best practices that your DevOps database solution should meet in order for you to get the most out of your investment.

### BEST PRACTICE: TRACK DATABASE CODE WITH APPLICATION CODE

Just like application software changes are tracked in source control, to understand the evolution of the database, it's equally important to track database changes with a source code control (SCC) solution as well. Choose a DevOps database solution that allows you to use your existing application SCC solution, such as Subversion, Git, Microsoft TFS, etc., instead of a solution that implements a separate SCC uniquely for the database or a solution that simply cannot track database changes in your SCC system out of the box. The goal is to minimize changes to the developer workflow without taking on a large integration or extension project.

By ensuring that the database automation solution integrates out-of-the-box with the same SCC system used for application code, developers can keep their existing workflow. This reduces the complexity, cost, and maintenance associated with having a separate SCC system in place for just the database.

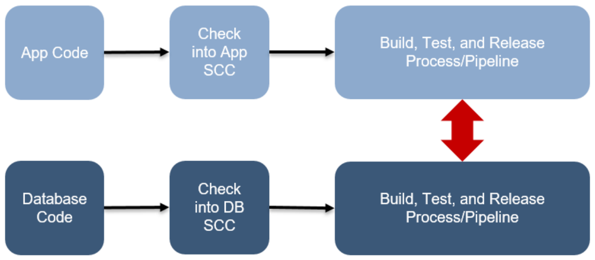
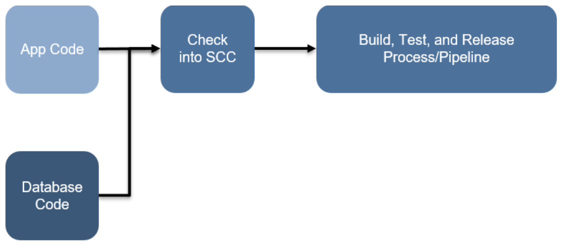

App Code
Check into App SCC
Build, Test, and Release Process/Pipeline
Database Code
Check into DB SCC
Build, Test, and Release Process/Pipeline

Diagram: It's counterproductive to have a separate source code control solution for the database code that is independent of the system used for application code. It means that effort has to be duplicated (code needs to be checked in two times to two different systems). It also means that database changes that support application features can drift and get out of sync as they flow through separate processes, adding to confusion and error while impeding velocity.



It's very efficient to check both application code and database code into a single source code control solution. There is no disruption to the development process, and it's much easier to keep application and corresponding database changes synchronized through the build, test, and release process/pipeline, as things can always be referenced against a single commit.

**Tip**: Be wary of solutions that have a separate SCC system in place for the database. With two SCC systems, either two people will follow two separate processes or one person will end up doing two duplicate tasks to achieve a goal. Coupled with the additional costs for integration and maintenance, adding another SCC solution just for the database is a poor investment that fails to eliminate error and improve visibility.

## BEST PRACTICE: RULE DEFINITION AND AUTOMATED ENFORCEMENT FOR PERFORMANCE AND SECURITY

Businesses are keen to minimize risk, cost, and downtime. A firm can go out of business if a critical application slows to a crawl, fails entirely, or worse, if production data is lost or becomes available to competitors or attackers. To maintain system performance and to protect their valuable data, organizations typically have a lot of rules, best practices, standards, and conventions governing the database. Currently, DBAs serve as a manual safeguard and end up reviewing any and all changes that need to be made to the database to ensure they meet all the organizational standards, rules, and best practices.

To eliminate the bottleneck of this manual process, it becomes necessary to automate the very necessary, yet mundane efforts of DBAs. This way, database changes can move at the speed of application changes and DBAs can be freed to address higher value projects such as performance improvements, upgrades, etc. Look for a solution that allows appropriate granularity in rule definition and enforcement. Ideally, this means a solution with an object model that can enforce changes at an individual changeset level so that rules such as "having a foreign key that points to a primary key that does not exist" can be easily defined and enforced.

**Tip**: Be wary of solutions that simply rely on regular expressions for rule definition. Relying solely on regular expressions makes it nearly impossible to manage simple structural standards. Instead, with regular expressions, a rule will have to be created to reject any changes relating to specific names or keywords and create a lot more review cycles and manual intervention for DBAs.

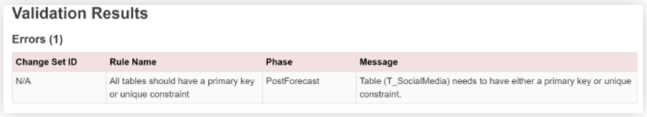
```
rule "All tables should have a primary key or unique
constraint"
        when
                    $db_model_container : ModelContainer( )
        then
                    String errorMessage = "";
                    for (Schema schema : $db_model_container.
getNewModel().getSchemas()) {
                                for (Table table : schema.
getTables()) {
                                            if (!table.getName().
toUpperCase().equals("DATABASECHANGELOGLOCK") && !table.
getName().toUpperCase().equals("DATABASECHANGELOG")) {
                                                        if ((table.
getPkConstraint() == null) && table.getUniqueConstraints().
isEmpty()) {

errorMessage += "Table (" + table.getName() + ") needs to
have either a primary key or unique constraint.<br/>\n";
                                                        }
                                            }
                                }
                    }
                    if (errorMessage != "") {
                                insert(new Response(ResponseType.
FAIL, errorMessage, drools.getRule().getName()));
                    }
end
```

This is an example rule definition done in Drools that can be consumed by Datical's Rules Engine. Datical is one of the commercially available solutions for database release automation that supports rule definition against objects, as recommended by the best practice.

Beyond rules that can be written against database objects at a changeset level, look for tools that can flag destructive changes before they happen. It's common for DBAs to manually check for things like, "Is there data in the column that this SQL script is trying to drop?" A tool that allows for the automation of such checks will save even more time and will better align the pace of database changes with application code changes.

**Tip**: SQL cannot simply be evaluated in isolation; it has to be evaluated in the context of the target database state. As such, look for products that can ingest and simulate changes against the target database state to truly free DBAs from the manual process and speed up the database release process.


Validation Results

Errors (1)

| Change Set ID | Rule Name | Phase | Message |
|---|---|---|---|
| N/A | All tables should have a primary key or unique constraint | PostForecast | Table (T_SocialMedia) needs to have either a primary key or unique constraint. |

This is an excerpt from a report generated by Datical DB, which has found some SQL that was checked in by a developer that violates the rule that tables should have a primary key or unique constraint. By automating and enforcing rules, these tools make it easy to spot issues in SQL change scripts and facilitate quicker remediation.

## BEST PRACTICE: ENABLE DEVELOPER SELF-SERVE AND PROVIDE IMMEDIATE FEEDBACK

Another fundamental tenant in addressing the database bottleneck is to enable developers to self-serve. Today, developers need to wait for days to hear back from a DBA about whether their change is valid or not and then repeat the cycle of waiting every time rework is necessary. In many organizations, the SLA between the DBA team and development team is defined in days or weeks. This means that a database change submitted by a developer at the start of a sprint may end up getting rejected at the very end of the sprint or possibly even after the sprint is completed, entirely throwing a release off track. The manual database change review process breaks DevOps and is simply unsustainable.

This is another issue that can be addressed with automation and Continuous Integration. It's a manual and tedious effort to involve DBAs in every change that a developer wants to make. Building on the previous best practice about defining rules and automating enforcement, this best practice is to look for a solution that can automatically evaluate the changes that developers check into SCC to provide feedback in minutes instead of the usual days or weeks it currently takes for a manual DBA review.  As a bonus, look for tools that integrate with build systems, such as Jenkins or in-house solutions, and which break the build if the database code doesn't pass validation.

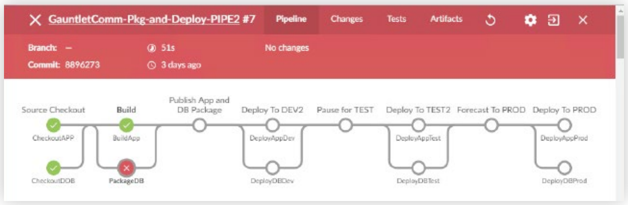
As a bonus, choose tools that further enhance feedback and developer self-serve. As an example, Datical integrates with Delphix so developers can get on-demand copies of masked production data. With better feedback and self-serve, the team can deliver more quickly and with less risk – especially since everyone can operate with an understanding of the live production environment.

**Tip**: Avoid introducing process changes that lock the database down further. In order to realize an improvement in database change velocity, it's necessary to allow developers to retain the workflow for application changes, where there can be simultaneous development against a given feature area. Merge conflicts are identified when pushing to source code, and functional issues that result in build failure or automated test failure are returned to developers for remediation. The database workflow needs to be able to mirror this, with checks running when code is checked into a source code repository and any failures getting immediately reported back to the developer.

Once the DBA team codifies organizational policies and checks into rules, the best database release automation tools allow developers to benefit directly from the automated rule enforcement. By integrating the automation with build tools, database changes that are checked in can immediately go through validation. If there is a bad change, it will break the build and developers will be immediately notified. This allows DBAs to be free from the constant development churn and focus their efforts on higher-priority projects instead of getting overrun and working overtime to keep up with development while allowing developers to self-serve and immediately get feedback on the database code they are checking in.

```
CREATE TABLE T_SocialMedia (
    [Socialid] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](200) NOT  NULL,
    [URL] [nvarchar](200)  NULL,
);
```

This is some sample SQL that is responsible for creating a new database table. With database release automation tools, when code like this is checked in and built, it results in build failures, as the automated rule checking can catch that this table doesn't have a primary key or unique constraint.

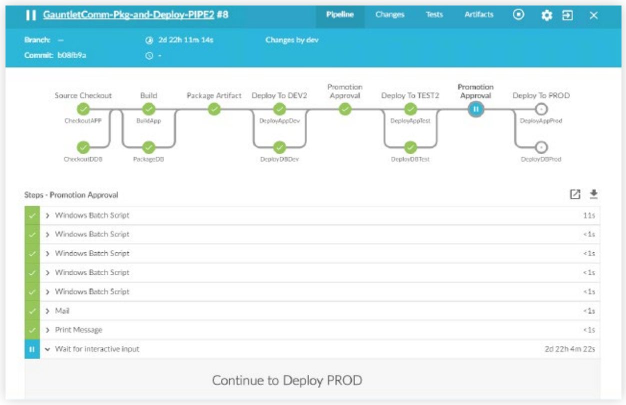

GauntletComm-Pkg-and-Deploy-PIPE2 #7

Branch: –
Commit: 8896273
51s
3 days ago
No changes

When tools like Datical are integrated with build automation tools like Jenkins, the build will fail if bad SQL code (like the example code above, creating the `T_SocialMedia` table) is checked into source code control and is included in the build pipeline. By integrating database code directly into the existing application release pipeline, any bad database changes can be immediately discovered and developers can be notified to make prompt fixes.

```
CREATE TABLE T_SocialMedia (
    [Socialid] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](200) NOT  NULL,
    [URL] [nvarchar](200)  NULL,
    CONSTRAINT PK_T_SocialMedia PRIMARY KEY (Socialid)
);
```

Once the build fails, the developer can make a fix and check in SQL that meets organizational standards and rules. In this case, it's simply a matter of not violating the rule that all tables should have a primary key or unique constraint by defining `Socialid` as a primary key.


GauntletComm-Pkg-and-Deploy-PIPE2 #8

Branch: –
Commit: b08fb9a
2d 22h 11m 14s
Changes by dev

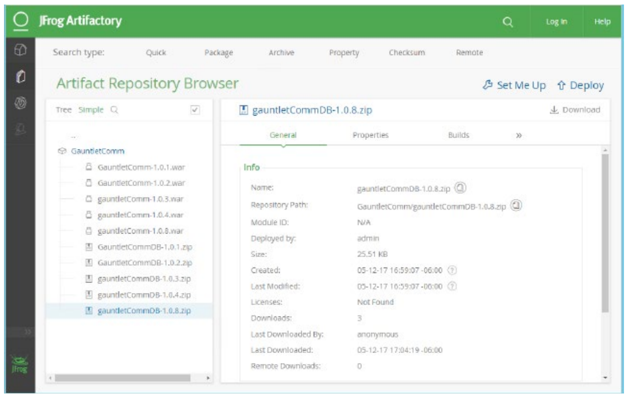
Steps - Promotion Approval

```
✓ › Windows Batch Script            11s
✓ › Windows Batch Script            <1s
✓ › Windows Batch Script            <1s
✓ › Windows Batch Script            <1s
✓ › Windows Batch Script            <1s
✓ › Mail                            <1s
✓ › Print Message                   <1s
II ⌄ Wait for interactive input     2d 22h 4m 22s
```

When the updated database code is checked in and a new build is kicked off, DevOps tools for the database can validate that no rules are violated and can allow the build to proceed. With integration with build tools like Jenkins, it's possible to see, in this case, that the change was successfully built and has been deployed to higher environments.

## BEST PRACTICE: BUILD ONCE, DEPLOY OFTEN

Just as with application code changes, it's important to follow the mantra of "build once, deploy many" with database changes, as well. If it requires custom or manual work to deploy a given database change into each environment along the pipeline, it becomes very difficult to isolate issues. Fundamentally, without building the database change into an artifact along with the application code, it becomes very difficult when things go wrong to tell if it is a bad database change that somehow worked in lower environments or if it is a valid change that is failing because there's something abnormal with the environment itself.


JFrog Artifactory
Log In
Help
Search type: Quick Package Archive Property Checksum Remote
Artifact Repository Browser
Set Me Up
Deploy
Tree Simple
Download
General Properties Builds

GauntletComm
- GauntletComm-1.0.1.war
- GauntletComm-1.0.2.war
- gauntletComm-1.0.3.war
- gauntletComm-1.0.4.war
- gauntletComm-1.0.8.war
- GauntletCommDB-1.0.1.zip
- GauntletCommDB-1.0.2.zip
- gauntletCommDB-1.0.3.zip
- gauntletCommDB-1.0.4.zip
- gauntletCommDB-1.0.8.zip

gauntletCommDB-1.0.8.zip

Info

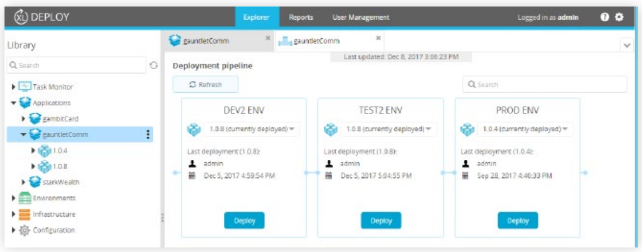
| | |
|---|---|
| Name: | gauntletCommDB-1.0.8.zip |
| Repository Path: | GauntletComm/gauntletCommDB-1.0.8.zip |
| Module ID: | N/A |
| Deployed by: | admin |
| Size: | 25.51 KB |
| Created: | 05-12-17 16:59:07 -06:00 |
| Last Modified: | 05-12-17 16:59:07 -06:00 |
| Licenses: | Not Found |
| Downloads: | 3 |
| Last Downloaded By: | anonymous |
| Last Downloaded: | 05-12-17 17:04:19 -06:00 |
| Remote Downloads: | 0 |

Tools like Datical integrate with artifact repository solutions such as JFrog Artifactory, and allow database code to be included in a single artifact that also contains the application code when the build is successful.

Solutions such as Datical allow database changes that have been checked into source code and that have passed the automated rule-checking to get built into an artifact along with application code. This way, the workflow automation and process can be maintained across the rest of the pipeline, and any issues can quickly be traced back to the environment or the offending code change.

## BEST PRACTICE: AUTOMATE THE DATABASE RELEASE

Beyond rules and artifacts, it is essential for the database release automation tool to actually automate the database change! However, before blindly deploying a change that has managed to pass through all the rules that have been created, look for tools that allow you to simulate the impact of a change before actually committing to the change. After all, there may be nuances to the environment in question and there may not be rules to safeguard against what might actually be a bad change.



Last updated: Dec 8, 2017 3:06:23 PM

| DEV2 ENV | TEST2 ENV | PROD ENV |
|---|---|---|
| 1.0.8 (currently deployed) | 1.0.8 (currently deployed) | 1.0.4 (currently deployed) |
| Last deployment (1.0.8): | Last deployment (1.0.8): | Last deployment (1.0.4): |
| admin | admin | admin |
| Dec 5, 2017 4:59:54 PM | Dec 5, 2017 5:04:55 PM | Sep 28, 2017 4:40:33 PM |

ToolsTools like Datical integrate with solutions like Jenkins, IBM Urban Code, CA Release Automation, and XLDeploy from XebiaLabs (shown above) in order to automate the actual deployment of the single artifact.

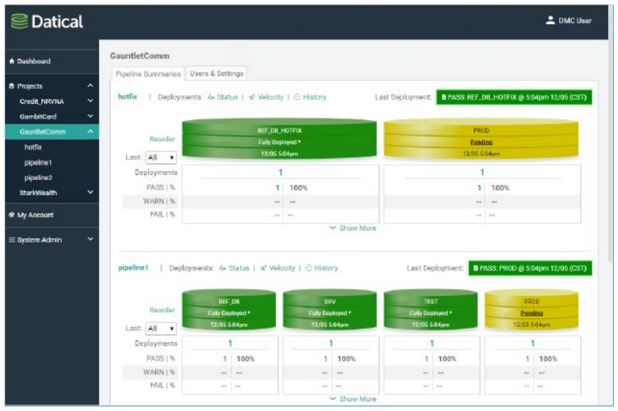
Database automation tools simulate changes to the database and compare the simulated database against the original. These tools also allow the rules used to validate changes to be selected based on environment. Lastly, by integrating with the rest of the application release automation pipeline, the best tools increase the overall return and value of the entire Continuous Integration and Continuous Delivery toolchain.

**Tip**: Be wary of tools that still keep database release automation a separate process from application automation. This can lead to the database and the application getting out of sync, and means that there is duplication in process or people during release. Make sure that the tools you select or augment integrate with your existing application release automation tooling.

## BEST PRACTICE: KNOW THE STATE OF EACH DATABASE

Both for operational sanity and for regulatory compliance, it's necessary for organizations to clearly understand the state of each of their databases — and certainly at least the production database! Look for a database release automation solution that provides visibility into state of each database in each environment. Look for a solution that can quickly answer, "What changes have been applied to this database?" or "What is the difference between the application database in the staging environment and the production environment?"

With more rapid release cadences and more people involved in the release process, it becomes increasingly more difficult to answer exactly what changes have and have not been applied to a database in a given environment. To avoid nasty surprises, to pass audits, and to ensure better uptime, it's essential to have a complete understanding of the state of each database.



hotfix — Last Deployment: PASS: REF_DB_HOTFIX @ 5:04pm 12/05 (CST)

| | REF_DB_HOTFIX<br>Fully Deployed *<br>12/05 5:04pm | | PROD<br>Pending<br>12/05 5:04pm | |
|---|---|---|---|---|
| Deployments | 1 | | 1 | |
| PASS \| % | 1 | 100% | 1 | 100% |
| WARN \| % | -- | -- | -- | -- |
| FAIL \| % | -- | -- | -- | -- |

pipeline1 — Last Deployment: PASS: PROD @ 5:04pm 12/05 (CST)

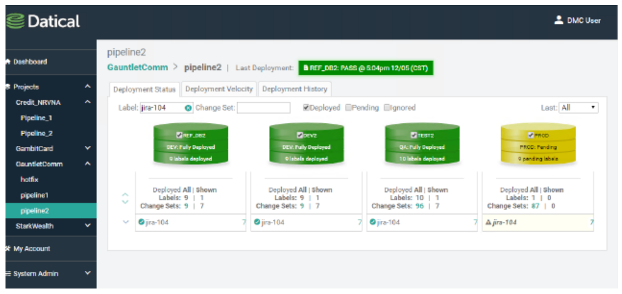
| | REF_DB<br>Fully Deployed *<br>12/05 5:04pm | | DEV<br>Fully Deployed *<br>12/05 5:04pm | | TEST<br>Fully Deployed *<br>12/05 5:04pm | | PROD<br>Pending<br>12/05 5:04pm | |
|---|---|---|---|---|---|---|---|---|
| Deployments | 1 | | 1 | | 1 | | 1 | |
| PASS \| % | 1 | 100% | 1 | 100% | 1 | 100% | 1 | 100% |
| WARN \| % | -- | -- | -- | -- | -- | -- | -- | -- |
| FAIL \| % | -- | -- | -- | -- | -- | -- | -- | -- |

Tools like Datical provide a dashboard view that summarizes the state of each database environment. Any environments that do not have the latest changes can be quickly identified, and teams can quickly drill into what outstanding changes have been made to each environment, regardless of the release pipeline that the environment belongs to.

### BEST PRACTICE: APPLICATION/DATABASE SYNCHRONIZATION AND FLEXIBLE FEATURE DEPLOYMENT

For a variety of reasons, it is common for an organization to revise the release plan to only include a subset of the features originally planned. As such, any database release automation solution needs to be able to accommodate for the inevitable feature churn that is a reality in high velocity Agile software development teams. Look for solutions that allow database changes corresponding to specific features and releases to be labelled accordingly, with mechanisms to ignore or unignore labels. The best tools integrate with ticketing systems such as JIRA to enable traceability from development all the way through the build and deployment pipeline.

Branch-based development, which is increasingly common across most enterprises, is much easier when the database release automation solution ensures that database changes can remain synchronized with application changes and flow in lockstep with application changes from development through production. Confusion around the question, "What application feature does this particular database change have to do with?" can be entirely eliminated. Release quality and is improved, as any unnecessary database changes that might impact the production version of the application can be left behind, along with corresponding application changes in lower environments until the feature set is ready for promotion to higher environments.

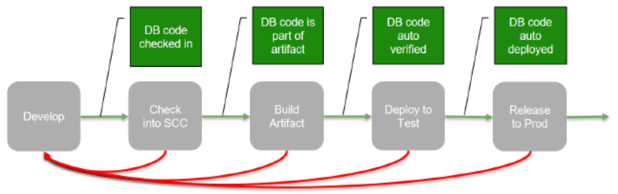


| | REF_DB2 | DEV2 | TEST2 | PROD |
|---|---|---|---|---|
| | Deployed All \| Shown | Deployed All \| Shown | Deployed All \| Shown | Deployed All \| Shown |
| Labels: | 9 \| 1 | 9 \| 1 | 10 \| 1 | 1 \| 0 |
| Change Sets: | 9 \| 7 | 9 \| 7 | 96 \| 7 | 87 \| 0 |
| | jira-104 7 | jira-104 7 | jira-104 7 | jira-104 7 |

Tools like Datical allow immediate visibility in specific feature sets by integrating with ticketing systems like JIRA and providing a web UI to understand where specific changes have been made. In this particular example, the change JIRA-104 has been deployed to all environments except Production. This ability to quickly understand the state of each database and know what has and has not been deployed can expedite troubleshooting and increase both the stability and the quality of releases.

## CONCLUSION

As teams continue to improve the throughput and velocity of application changes, the database is increasingly the critical bottleneck. As long as database changes remain a manual process, no increase in DBA headcount can scale the manual process to keep up with application updates. It's necessary for teams to adopt true database automation and to treat database code just like application code in order to eliminate the database bottleneck. Given that the database holds valuable state, it's equally necessary that any automation tools for the database are equipped with sufficient safeguards to avoid data loss, application downtime, or security issues from occurring when an update is pushed live in production.



With proper database release automation, the database is no longer a bottleneck. Database changes can flow in sync with application changes through the release pipeline.

In considering a database change and release automation solution, keep in mind the best practices outlined to ensure the best return on investment. The best database automation solution will seamlessly fit into an existing application release automation toolchain and will substantially increase the overall value of the existing software release pipeline.

### ABOUT THE AUTHOR

**SANJAY CHALLA** is a senior product marketing manager with over six years of experience in enterprise software. With previous experience in both product marketing and product management, Sanjay has a deep understanding of modern software engineering tools and methodologies.

DZone BROUGHT TO YOU IN PARTNERSHIP WITH Datical

Refcardz Feedback Welcome: refcardz@dzone.com
Sponsorship Opportunities: sales@dzone.com
DZone, Inc. 150 Preston Executive Dr. Cary, NC 27513
888.678.0399 - 919.678.0300