Lessons learned implementing ChatOps (DevOps + messaging)

Corey Hulen / Co-founder and CTO, Mattermost, Inc.  
Aug 2019
What is ChatOps?
ChatOps is a collaborative, conversation-centric way of working that brings people, discussions, bots, tools and files together in one central location.
Benefits of ChatOps

- Shorten the feedback loop (Faster time to response)
- It is a shared console or command line
- It captures the history
- All the required people in one place
- Mean time to restore drops dramatically
- It’s a methodology not a technology
Ideal World of ChatOps

**Reality - most are not that sophisticated, but even the smallest steps add value!**
That’s great, but how do we get there?
Step 1 - Start Small, do it manually for a bit

- Are you a chat based organization?
- Start by getting all the humans into a channel based chat system
- Channel base communication is key to incident response
- It’s a shared screen - think of it like a cooperative board game
- It captures the history of everything everyone has tried
- Don’t be afraid to over add people to start
- Your team has a lot of esoteric knowledge that can be valuable while troubleshooting
Team Coop Solving a Problem Manually

Sudheer Timmaraju 1:27 AM

Hey @patniharshit. Screenshot here does not capture error log. Mind checking your log again about the errors and posting it?

It says the build failed but does not capture the reason of the failure to help you out.

patniharshit 1:45 AM

0 info it worked if it ends with ok
1 verbose cli [ '/usr/local/bin/node', '/usr/local/bin/npm', 'run', 'build' ]
2 info using npm@6.4.1
3 info using node@v11.0.0
4 verbose run-script [ 'prebuild', 'build', 'postbuild' ]
5 info lifecycle mattermost-webapp@0.0.1~prebuild: mattermost-webapp@0.0.1
6 info lifecycle mattermost-webapp@0.0.1~build: mattermost-webapp@0.0.1
7 verbose lifecycle mattermost-webapp@0.0.1~build: unsafe-perm in lifecycle true
8 verbose lifecycle mattermost-webapp@0.0.1~build: PATH: /usr/local/lib/node_modules/npm/node_modules/npm-lifecycle/node-gyp-bin:/home/patniharshit/go/src/github.com/mattermost/mattermost-webapp/node_modules/.bin:/usr/local/cuda-8.0/bin:/home/patniharshit/bin:/home/patniharshit/.local/bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/usr/games:/usr/local/games:/snap/bin:$HOME/ Documents/Nim/bin:/usr/local/go/bin:/home/patniharshit/go/bin:/usr/local/go/bin
9 verbose lifecycle mattermost-webapp@0.0.1~build: CWD: /home/patniharshit/go/src/github.com/mattermost/mattermost-webapp
10 silly lifecycle mattermost-webapp@0.0.1~build: Args: [ '-c',
10 silly lifecycle 'cross-env NODE_ENV=production webpack --display-error-details --verbose' ]
Step 2 - Automate the Simple Things

- You don’t need to start with an uber powerful bot that solves every problem
- Automate simple alerts piping them into a channel
- Create specific channels for the different alerts
  - Security
  - Performance
  - Build System
  - War rooms
- Use simple webhooks to get you started
- Watch your team cooperatively respond
  - How often does the left hand not know what the right hand is doing?
Solving a problem from a simple Webhook

ALARM: "corp-mattermost-CPU-Utilization" in US East (N. Virginia)

#aws-notification-alert on Sat, 3 Nov 2018 18:45:57 +0000

You are receiving this email because your Amazon CloudWatch Alarm "corp-mattermost-CPU-Utilization" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 datapoint [82.461844956938 (03/11/18 18:40:00)] was greater than or equal to the threshold (50.0)." at "Saturday 03 November, 2018 18:45:57 UTC".

View this alarm in the AWS Management Console: https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#s=Alarms&alarm=corp-mattermost-CPU-Utilization

Alarm Details:
* 'Unit' not specified

Carlos Panato 12:04 PM
@Martin Kraft

System 12:04 PM
@Martin Kraft added to the channel by @Carlos Panato.

Martin Kraft 12:07 PM
Commented on alert's message: ALARM: "corp-mattermost-CPU-Utilization" in US East (N. Virginia) #aws-notification-alert on Sat, 3 Nov 2018...

I am doing a lot of requests right now.
Step 2 - Automate the Harder Things

- Starting adding advanced slash commands or alerting hooks
- Bring in Richer Data
- More than Screenshots
- Markdown Tables
- File attachments for logs
- Markdown formatting
Example of Rich Markdown Table

Loadtest Results

Score: 71.48 (+24.66, relative to baseline)

The score is the average of the 95th percentile, median and interquartile ranges in the routes below.

Routes

GET /channels/[channel id]/members

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>Actual</th>
<th>Delta</th>
<th>Delta %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hits</td>
<td>1515</td>
<td>1455</td>
<td>-60</td>
<td>-3.96%</td>
</tr>
<tr>
<td>Error Rate</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Mean Response Time</td>
<td>4.91ms</td>
<td>2.69ms</td>
<td>-2.22ms</td>
<td>-45.16%</td>
</tr>
<tr>
<td>Median Response Time</td>
<td>4.00ms</td>
<td>2.00ms</td>
<td>-2.00ms</td>
<td>-50.00%</td>
</tr>
<tr>
<td>95th Percentile</td>
<td>8.00ms</td>
<td>5.00ms</td>
<td>-3.00ms</td>
<td>-37.50%</td>
</tr>
</tbody>
</table>
Example of Rich messages and log files

jira bot 12:31 PM

Harrison Healey created Bug MM-13014

Web: Sometimes tries to load posts before current channel change
Repro steps (may be inconsistent):
1) Open Mattermost in a DM channel
2) Observe the JS console and server logs

Observed: The client requests /api/v4/channels/undefined goes out shortly after with the correct channel ID (second request)

Corey Hulen 1:47 PM

Attaching nginx log for bug report

ngxin.txt

TXT 7KB
Step 3 - Start Automating Command and Control

- Start off simple with outgoing webhooks and slash commands
- Automate the things you find your teammates constantly bugging someone to do
- Allow self provisioning by most anyone
- Trust the automation to keep them on the rails (They shouldn’t be able to screw it up)

Examples
  - Setup adhoc testing environments
  - Setup load testing environments
  - Query detailed analytics systems

Automate deploys to production, why?
  - Nice history of what went wrong, anyone/everyone can help out
Matterbuild example

cpanato 21:18
ok
for 5.5: pr-mattermost-server-9799
for master: pr-mattermost-server-9798

/mb cut 5.5.0-rc1

Matterbuild BOT 11:12
Matterbuild
Cut Release
Release 5.5.0-rc1 is on the way.

Mattermost Release BOT 11:15
Release 5.5.0-rc1 has started building.
Matterbuild example cont.

Mattermost Release

**Release 5.5.0-rc1 Download Links**

**Enterprise Edition**
- Link: [https://releases.mattermost.com/5.5.0-rc1/mattermost-5.5.0-rc1-linux-amd64.tar.gz](https://releases.mattermost.com/5.5.0-rc1/mattermost-5.5.0-rc1-linux-amd64.tar.gz)
- MD5 Hash: f8d9e856158a3d33489b36e4735583ee
- SHA-1 Hash: eba3ba8610d4d867e452d60b916f55156692a71
- SHA-256 Hash: a3c23d86dc33491b94117ae00882def13e036956955896f1dc83ed199d2a29

**Team Edition**
- Link: [https://releases.mattermost.com/5.5.0-rc1/mattermost-team-5.5.0-rc1-linux-amd64.tar.gz](https://releases.mattermost.com/5.5.0-rc1/mattermost-team-5.5.0-rc1-linux-amd64.tar.gz)
- MD5 Hash: 55b4d75e89736c6995486979bc1e7a8
- SHA-1 Hash: 5e08df2d3d3f446a275c978ab1f43f8c1b453b7572
- SHA-256 Hash: 3b06d2b9317e1e6f212121414f95e84f171ee081076e817513dc19ed9e96

**Testing Server**
- [https://rtesting.redhat.com](https://rtesting.redhat.com)

**GPG Signatures**
- Key: A1831D46F0F3A1D802CF2D44F82C31744774B28
- EE Signature: [https://releases.mattermost.com/5.5.0-rc1/mattermost-5.5.0-rc1-linux-amd64.tar.gz.sig](https://releases.mattermost.com/5.5.0-rc1/mattermost-5.5.0-rc1-linux-amd64.tar.gz.sig)
- TE Signature: [https://releases.mattermost.com/5.5.0-rc1/mattermost-team-5.5.0-rc1-linux-amd64.tar.gz.sig](https://releases.mattermost.com/5.5.0-rc1/mattermost-team-5.5.0-rc1-linux-amd64.tar.gz.sig)
Matterbuild example cont.

Control of the build system through MM slash commands!

**Usage:**
matterbuild [command]

**Available Commands:**
cut: Cut a release of Mattermost

cutstatus: Check the status of the Cut Release Job

getpoozie: Check the branches set in the Translation Server

help: Help about any command

loadtest: Create a kubernetes cluster to loadtest a branch or pr.

lockpoozie: Lock the Translation server for a particular release Branch

merge: Merge the specified release branch to master and create the pull request

runjob: Run a job on Jenkins.

seeconf: Dump the configuration of a build job.

setci: Set the branch target for the CI servers.

setprerelease: Set the target for pre-release.

Use "matterbuild [command] --help" for more information about a command.
Step 4 - Bot It Up?

- Take all the great work you’ve done previous steps and start converting it to a more human interface
- I don’t know about you, but I have a hard time remembering slash commands and all their parameters (who reads the help??)
- Give it some character, make it fun!
  - Make it angry
  - Or overly happy
  - Humanize it to reach a larger audience
  - Gives you someone to blame. “The angry bot did it, not me!”
- Use existing bot interfaces like Hubot
  - Jenkins
  - Show server logs for various test servers
  - Show Grafana reports
Hubot Example

Corey Hulen  9:34 AM
hubot lets ship it

hubot  9:34 AM
✓ http://shipitsquirrel.github.io/images/ship%20it%20squirrel.png
Step 5 (Optional) - Build Rich Plugins and Bots

- Utilize the rich plugin framework to build deep integration widgets (Mattermost Only)
- Both frontend and backend plugins
- Intercept messages before they are entered into chat and after they are entered into chat
Github Integration
Github Integration Cont.
Jira Integration

George Goldberg 1:18 PM

Sounds good to me.

Jesse Hallam 10:55 AM

Came across a surprise in our use of Editor objects and the spread operator. Some of our newer Redux actions use this paradigm and look like this:

```javascript
let data;
try {
    data = await Client4.patchMe(user);
} catch (error) {
    dispatch({
        type: UserTypes.UPDATE_ME_FAILURE, error
    }, getState);
    return (error);
}
```

When an error occurs, the returned object contains an error field with an instance of ClientError. Sometime down the line, we might do something like this (using our old action styles):

```javascript
export async function updateUser(user, success, error) {
    const { data, error: err } = await UserActions.updateMe(user)(dispatch, getState);
}
```
Jira Integration Cont.

Create Jira Ticket

- **Project**: Customer Requests
- **Issue Type**: Feature Request
- **Summary**
- **Description**
  
  Came across a surprise in our use of `Error` objects and the spread operator. Some of our newer Redux actions leverage the try/catch paradigm and look like this:

  ```javascript
  (function() {
    let data;
    try {
      ...other logic...
    } catch (error) {
      // Handle error
    }
  })();
  ```

  Notice the `error` callback passing `...err`. If you look at the error object being passed (for a different instance than this one), you'll see that the `message` attribute is a property on the root `Error` object (greyed out, alongside the stack, since it's part of the error object).
Step 6 - Nirvana, Bots talking to Bots :) 

- Some customers are generating 50% of their traffic from bots
- Some customers have a war room that looks like reading the matrix
- Message scrolling faster than a human can read, why?
- Bots are watching the war room, pulling data out into other channels where humans process the information then bots put the info back in the war room channel
- War room is acting like an app message bus at this point, but it gives everyone a common interface
Questions?
How to reach me?

- Twitter: @corey_hulen
- Email: corey@hulen.com
- Mattermost: @corey on https://community.mattermost.com