

Building Cloud Native Microservices with Liberty and Node.js: A Product Development Journey

Tom Banks: IBM Offering Manager
and Technical Evangelist

InterConnect 2017



Please note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Goals of this session

Provide insights into what it means to develop cloud native microservices

Discuss the tools used throughout the process

Talk about developing with WebSphere Liberty and Node.js

Use a reference project developed over the last year to show real examples

What does cloud native really mean?

- Microservices
- Containers from development to production
- Continuous Delivery/Continuous Integration
- Deploy to the cloud (or on premise) using consistent deployment methods
- Portability between clouds built on the same technologies

An **application architecture** designed to leverage the **strengths** and accommodate the **challenges** of a **standardized** cloud environment, including concepts such as **elastic** scaling, **immutable** deployment, **disposable** instances, and **less predictable infrastructure**.



IBM Voice Gateway

Customer value

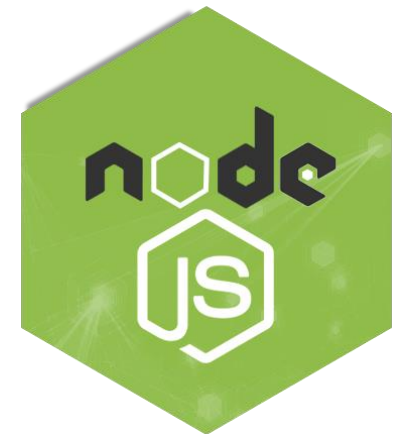
Based on customer demand, with customers able to provide continuous feedback, testing, and new requirements.

Agile development

Small, agile development team with minimal process, and the ability to deliver value quickly and easily.

Use the best of IBM technology

To build a stable, scalable application we needed to leverage our existing technologies as a foundation to the new offering.





IBM Voice Gateway

Connect to Cloud

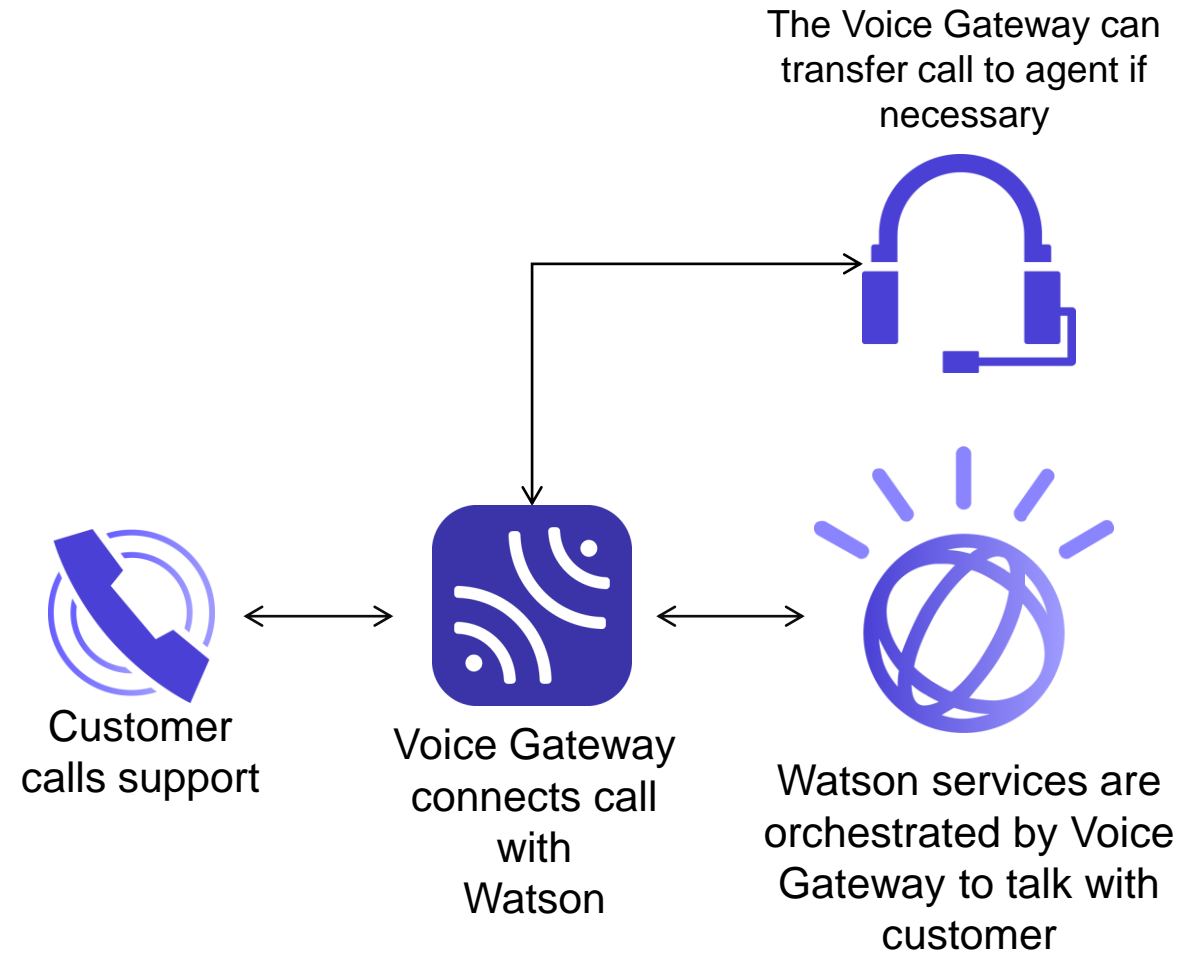
IBM Voice Gateway is a brand new connect to cloud offering that uses WebSphere Liberty to connect the telephone network to Watson's services in Bluemix

Bring Watson to your call center

Through connecting Watson Speech To Text, Conversation, and Text To Speech, to the phone network you can use Watson as a self-service customer service agent

Cloud native

The IBM Voice Gateway is a cloud native solution, comprising of two microservices delivered as Docker Images - this gives maximum flexibility in a cloud native world



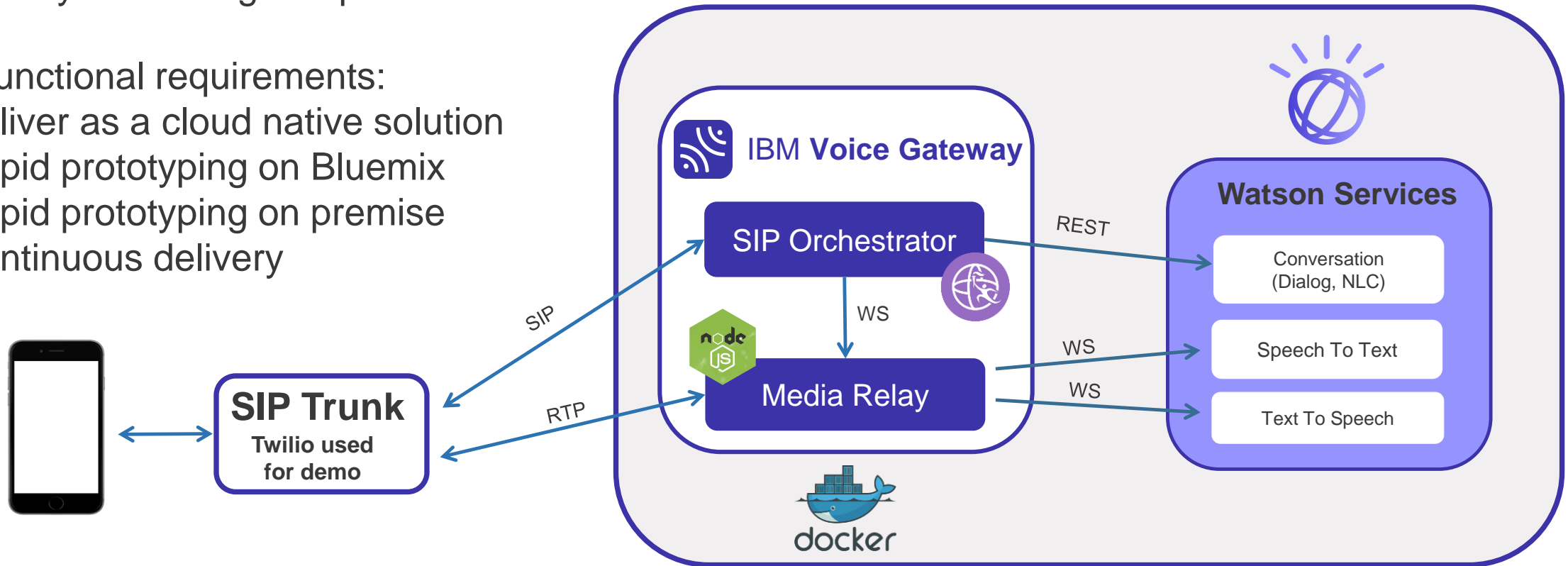
Reference Project Microservice Architecture

Project attributes

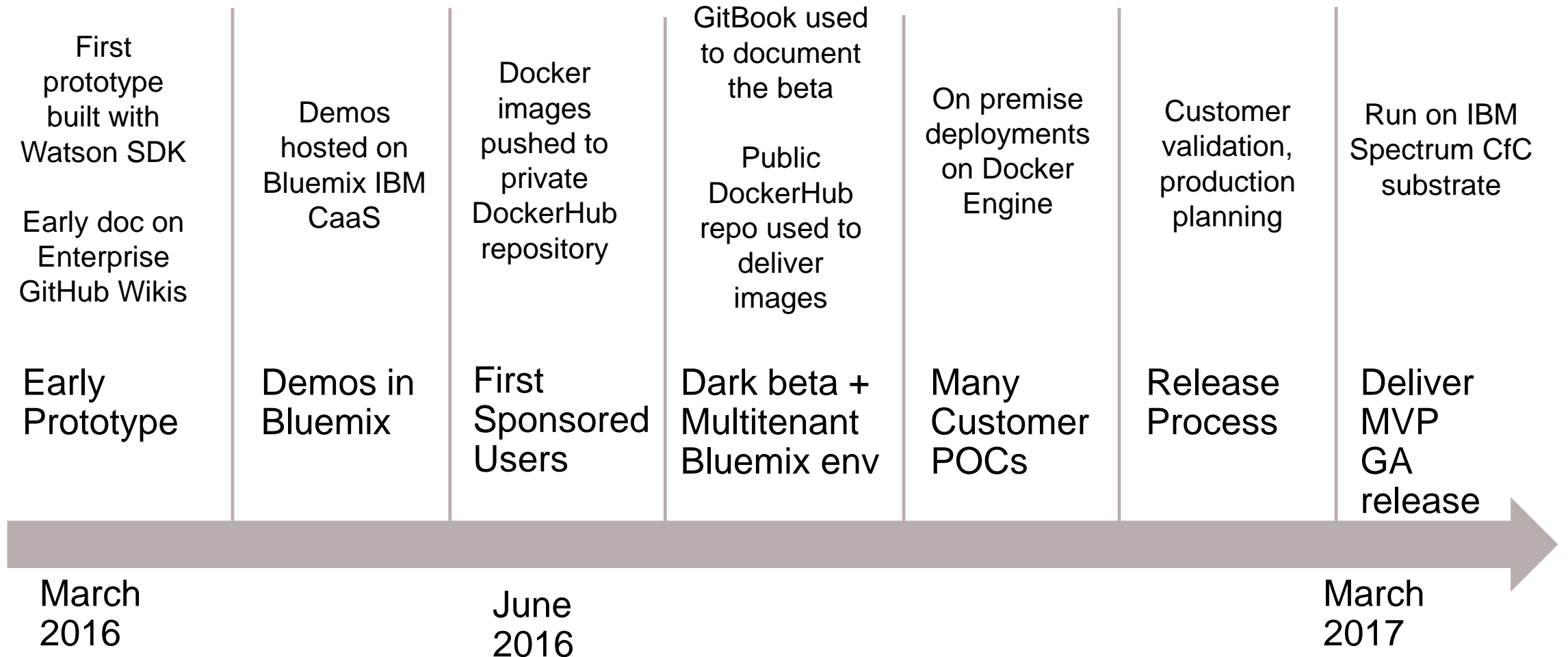
- ✓ Work as a 'startup' inside IBM
- ✓ Built by a small agile squad

Non-functional requirements:

- ✓ Deliver as a cloud native solution
- ✓ Rapid prototyping on Bluemix
- ✓ Rapid prototyping on premise
- ✓ Continuous delivery



Reference Project Timeline



Organizing your teams for cloud native development

Spotify engineering allowed for lots of autonomy

- Teams organized into squads that have the freedom to define what tools, processes and procedures work best for them
- Encourages a culture of innovation and continuous improvement
- Strips away bureaucracy

Impact on my team

- Developed our own CI/CD pipeline (based on new corporate tools)
- Developed many early prototypes
- Worked directly with early adopters which helped us quickly figure what was important and what wasn't
- Autonomy allowed for team to work directly with customers to define product

Working with Git and GitHub

Git is by far the best source control option for developing cloud native services:

- Integrates well with modern DevOps tools
- Extremely agile and flexible
- Public and private repositories

Source control for reference project managed by:

- Internal Enterprise GitHub repository for source
- External GitHub repository for sample scripts and beta documentation

Provided a way to share source, samples and wikis across your organization

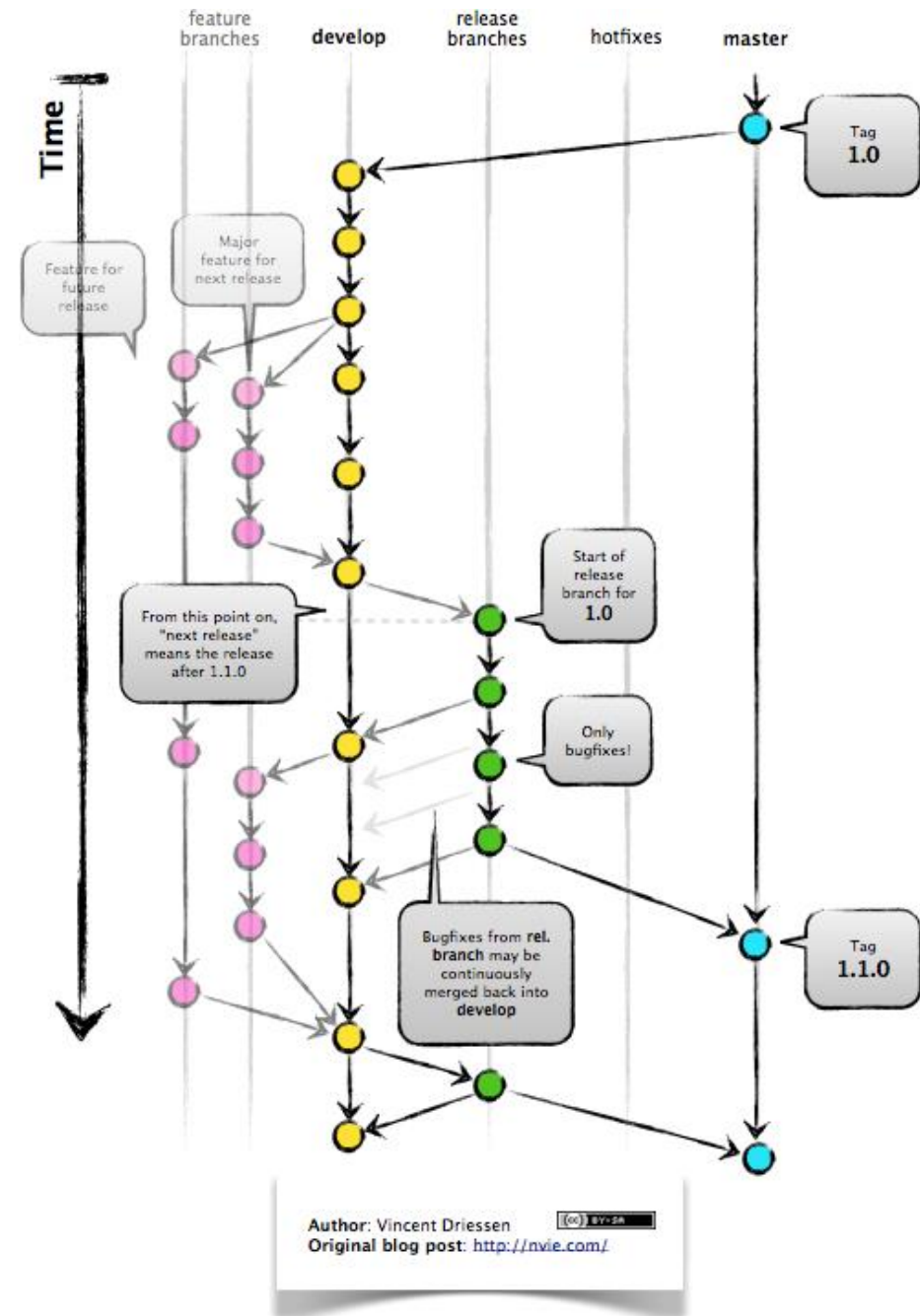
- No divisional restrictions
 - Enable anyone in the company to open an issue or request for enhancement
 - Cultivated an open source community inside of IBM
-
- Lessons learned
 - Agree early on team processes around the use of Git

Git Flow

Git Flow defines how your team develops and delivers code using git.

Lessons learned:

- How your team works with git is important to define up front.
- Git flow is fairly standard and lots of good examples exist.



Working with ZenHub

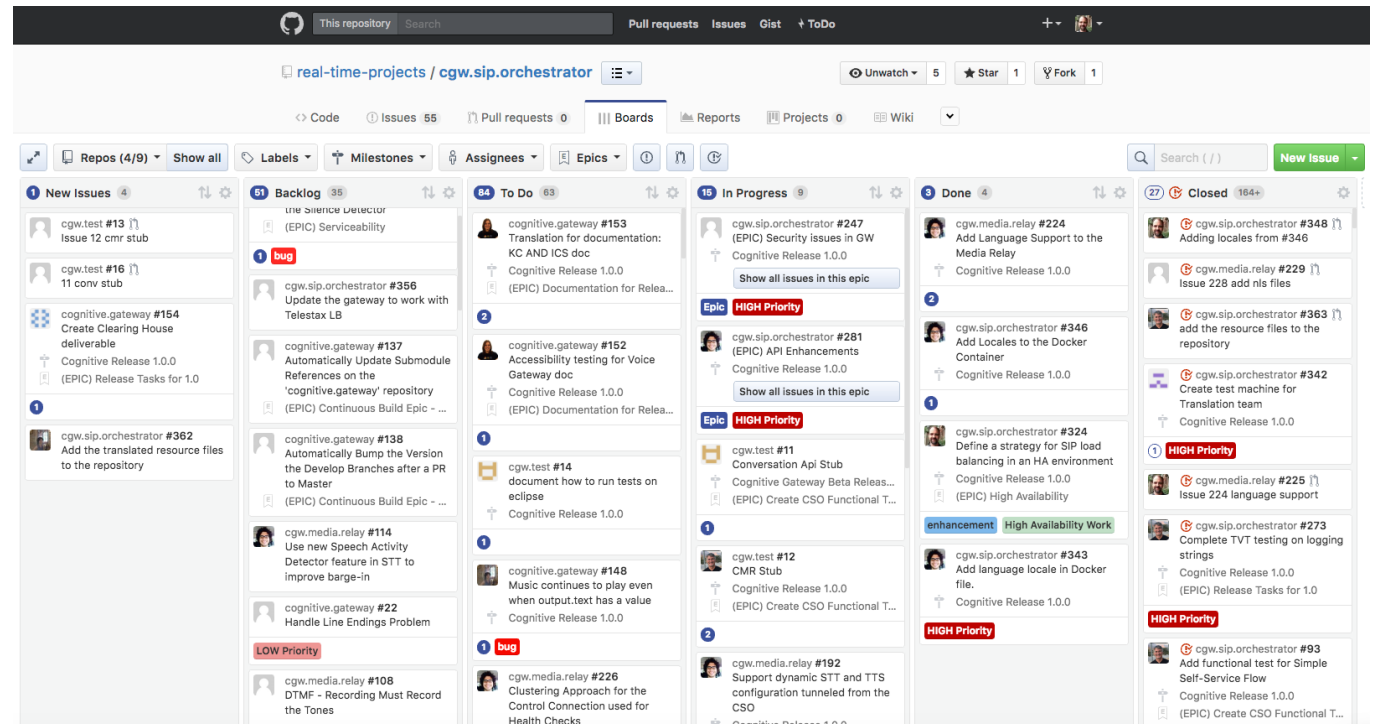
ZenHub is an enterprise-ready project management tool that adds features right into GitHub.

Features include:

- Epics, Milestones and Issues
- Burn down charts
- Issues pipeline

Lessons learned:

- Include issue number in branch name
- Epics do not span repositories



Working with Docker images

Early development with Docker

- ✓ Install Docker (Docker Engine, Docker for Mac, etc.)
- ✓ Create images locally with Docker build files
- ✓ Configuration through Docker environment variables

Docker compose:

- ✓ Tool for defining and running multi-container Docker applications.
- ✓ Provides a nice way to manage configuration of multiple containers from a single file.

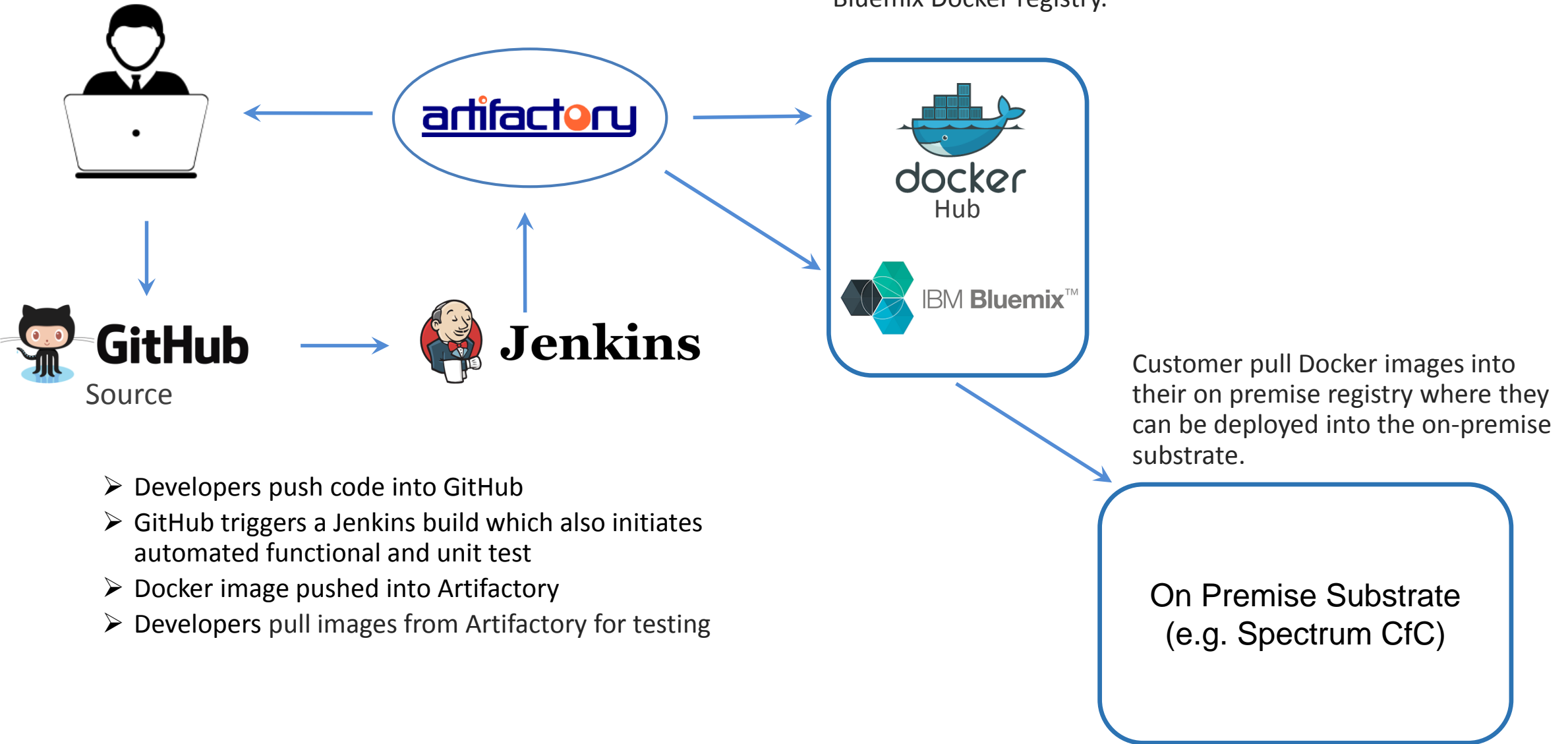
Internal Docker image distributions used Artifactory

- ✓ Pipeline pushes a new Docker image after every pull request
- ✓ Allowed team to make images available to internal customers
- ✓ Allowed team to quickly test new image builds
- ✓ Access strictly controlled by squad

External Docker Image Distributions

- DockerHub currently being used for public distribution of Docker images
 - Easy to setup CI/CD pipeline to push images
 - Easy for customers to pull images
- Started out with a private DockerHub repository
 - DockerHub access to images controlled by squad
- Later moved to public DockerHub repository for beta and release images
 - Accessible by anyone through pull requests

Our CI/CD Pipeline



- Developers push code into GitHub
- GitHub triggers a Jenkins build which also initiates automated functional and unit test
- Docker image pushed into Artifactory
- Developers pull images from Artifactory for testing

Jenkins Orchestrates Build and Test Automation

- Jenkins Pipeline defined by Jenkins file
 - Build automation script
 - Maintained with the source code of a repository
 - Gives developers the ability to modify or view the build script at will
 - Serves as a single source of truth
- Jenkins triggered by git pull requests and merges
- Jenkins builds utilize Gradle files checked into source projects















Jenkins

	checkout	build	push	acceptance tests
	7s	1min 2s	20s	4min 37s
#90 Mar 10 14:51 4 commits	8s	1min 19s	13s	6min 7s
#89 Mar 10 12:33 1 commits	7s	1min 20s	11s	6min 54s
#88 Mar 10 11:54 1 commits	8s	1min 25s	15s	7min 7s

Jenkins dashboard for one of our Git repositories





















 cgw.sip.orchestrator

Branches **Pull Requests**

S	W	Pull Request ↓	Last Success	Last Failure	Last Duration	Fav
		PR-369	15 hr - #27	37 min - #28	12 min	 
		PR-436	37 min - #7	2 hr 55 min - #5	16 min	 
		PR-438	37 min - #2	N/A	10 min	 

 cgw.sip.orchestrator

Branches Pull Requests

S	W	Branch ↓	Last Success	Last Failure	Last Duration	Fav
		develop	15 hr - #84	2 days 2 hr - #79	17 min	 
		master	1 mo 6 days - #13	3 mo 0 days - #11	6 min 15 sec	 
		performance-tweaking	9 days 19 hr - #1	N/A	1 min 38 sec	 
		release-0.3.0	1 mo 6 days - #4	1 mo 6 days - #3	5 min 58 sec	 
		release-0.3.1	1 day 2 hr - #1	N/A	26 min	 

Test Automation

- Unit and functional testing
 - Liberty based Microservice
 - Junit
 - Bash scripts (functional testing built in Java)
 - Node.js based Microservice
 - Mocha for unit and functional testing
- System automation test
 - Jenkins builds Docker images and executes bash script
 - Docker containers launched from bash scripts
 - System test code runs in Java



Cloud native development with WebSphere Liberty and Node.js

- Both runtimes are extremely lightweight (small footprint, start in seconds)
- Both runtimes already available as IBM provided Docker images
- Both runtimes integrate well with other Microservices

WebSphere Liberty

- Java
- Great for process intensive workloads
- Multi-threaded
- Java EE and SIP based workloads
- Great IDEs (Eclipse) and debuggers

Node.js

- JavaScript
- Great for networking workloads
- Single threaded
- Completely asynchronous
- Minimal context switching under load

Building a Liberty based application is simple using Docker

Docker file attributes

- Based on image from Bluemix
- Set Docker environment vars
- Copy feature (or war) to image

```
FROM registry.ng.bluemix.net/ibmliberty:webProfile7
```

```
MAINTAINER Brian Pulito <brian_pulito@us.ibm.com>
```

```
RUN apt-get update \  
    && apt-get -y install vim \  
        && apt-get clean \  
        && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

```
# Set the locale  
RUN locale-gen en_US.UTF-8  
ENV LANG en_US.UTF-8  
ENV LANGUAGE en_US:en  
ENV LC_ALL en_US.UTF-8
```

```
ENV SIP_PORT 5060
```

```
ENV LOG_LEVEL audit
```

```
ENV FEATURE cgw.sip.orchestrator.feature_0.2.0.esa
```

```
COPY ./servers/cgw-server/server.xml /config/
```

```
#Sets up the CognitiveSipOrchestrator Feature
```

```
COPY $FEATURE /tmp/
```

```
RUN installUtility install --acceptLicense /tmp/${FEATURE} && rm /tmp/${FEATURE}
```

```
# Add the plugins
```

```
COPY configDropins /config/configDropins
```

```
COPY dropins /config/dropins
```

```
# Add the Licenses
```

```
COPY licenses /licenses
```

```
EXPOSE $SIP_PORT
```

The Liberty server.xml and Docker env variables

Passing Docker env variables to your application:

- If var not set, \${xxx} passed to application

```
<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>usr:cgwSipOrchestrator-1.0</feature>
  </featureManager>

  <!-- Config for the WSG service -->
  <cognitiveGateway
    rtpRelayHost="${env.RTP_RELAY_HOST}"
    rtpRelaySecure="${env.RTP_RELAY_SECURE}"
    convWorkspaceID="${env.CONV_WORKSPACE_ID}"
    convEndpoint="${env.CONV_ENDPOINT}"
  />

  <sipEndpoint host="*" sipTCPPort="${env.SIP_PORT}" sipUDPPort="${env.SIP_PORT}" />
  <webContainer deferServletLoad="false"/>

  <logging traceSpecification="*=info:com.ibm.ws.cgw.*=${env.LOG_LEVEL}"/>
  <sipStack sentByHost="${env.SIP_HOST}"/>

</server>
```

Building a Node.js based application is simple using Docker

Docker file attributes

- Based on image from Bluemix
- Add npm modules
- Set Docker environment vars
- Startup node

```
FROM registry.ng.bluemix.net/ibmnode:latest
ENV NODE_ENV production

ADD . /cgw-media-relay
WORKDIR /cgw-media-relay

RUN apt-get update \
  && apt-get -y install \
  curl \
  vim \
  && apt-get clean \
  && npm install \
  && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

#
# Used to set port range
# These can only be done here in a Docker container, or else we can't expose right ports.
#
# expose 100 Ports

ENV RTP_UDP_PORT_RANGE="16384-16484"
ENV RTP_PORT=8080

EXPOSE $RTP_PORT
EXPOSE $RTP_UDP_PORT_RANGE/udp

CMD ["/cgw-media-relay/bin/rtp-relay"]
```

The importance of rapid prototyping in the cloud

- Great way to reach customers for early demos
- Provides a path to a true cloud service
- Feedback from early adopters was invaluable
 - Customer input drove feature/functions
- Lessons learned:
 - Be prepared for port scanners to disrupt your application (fix using white listing, mutual authentication, etc.)
 - Think multi-tenancy instead of a new env/space for every customer
 - Be prepared to spend a lot of time supporting POC environments for other people!

Working with IBM Container Service (ICS) in Bluemix

- ICS provides a way to quickly demo in a way that both internal and external users can easily access over public internet
- Process to deploy:
 1. Tag Docker images for your Bluemix repository.
 2. Push images to Bluemix
 3. Create containers
- See sample bash ICS deployment scripts here:

<https://github.com/WASdev/sample.voice.gateway>

- Lessons learned
 - Docker compose does not work with ICS (use env files instead)

Using Docker images to deliver on premise

Docker only release has its advantages:

- Develop, test and deliver code in the form of Docker images
- Deliver software through existing public repositories like DockerHub
- Makes it really easy for customers to update to new releases

Things to consider:

- Configuration through Docker environment variables
- Container orchestration
- Security for hybrid cloud access

Lessons learned

- Docker on premise is still not widely adopted in production (this is changing quickly)
- Leave extra time for hybrid connectivity issues (security, MPLS, etc.)

Becoming production ready

Container Orchestration

- Orchestration needed for cluster management:
 - Container scheduling, Auto-scaling, Rolling Updates, etc
- Options include Kubernetes, Docker Swarm, Mesos
 - Kubernetes has been the focus of the reference project
- Lessons learned:
 - Make sure the framework meets your load balancing requirements like session affinity and protocol support



IBM Spectrum Conductor for Containers



Open Source + Integration + Value Add



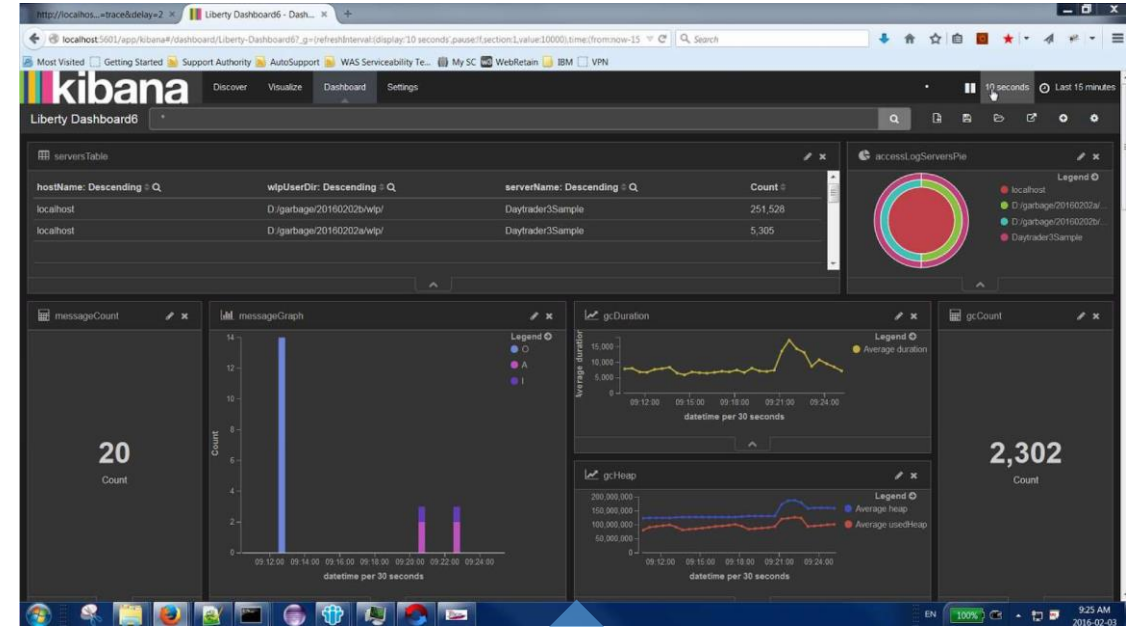
Deploying Docker images on CfC for production

- IBM Spectrum Conductor for Containers provides an IBM supported, Docker substrate to run non-managed containers.
 - Docker registry
 - Orchestration through Kubernetes
 - Management, monitoring, log aggregation, etc.
 - <https://www.ibm.com/developerworks/community/groups/service/html/communityoverview?communityUuid=fe25b4ef-ea6a-4d86-a629-6f87ccf4649e>

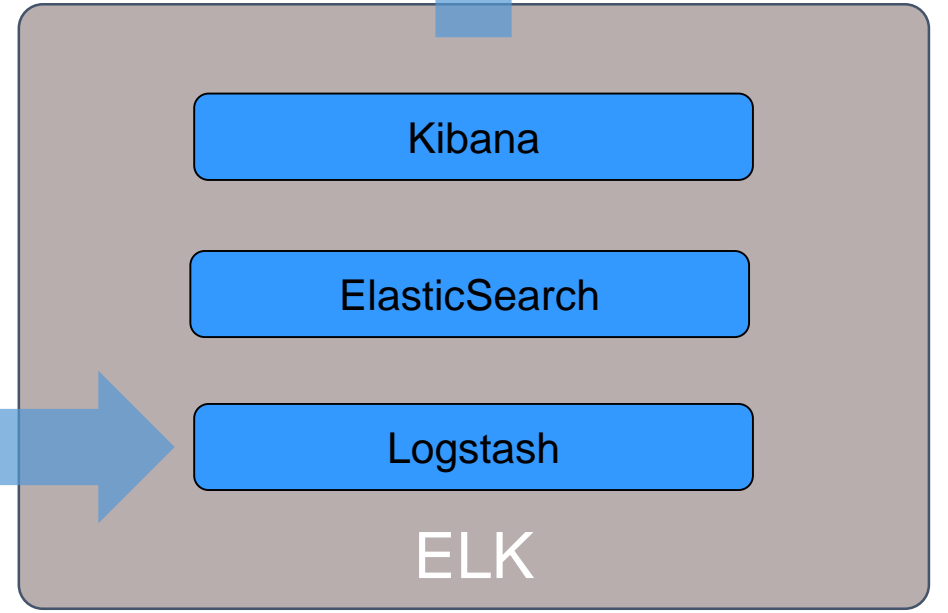
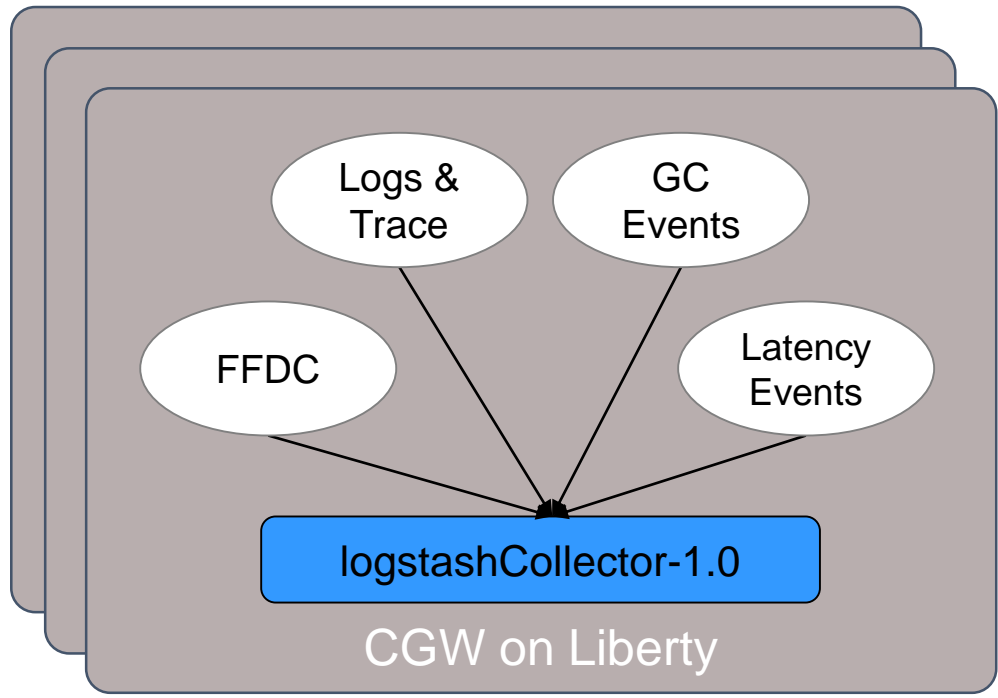
Log Analytics

Connect to Logmet/Bluemix or your own ELK

- Consolidate data from any servers that have access to your Logstash server



Browser



Reporting and Monitoring

On premise reporting

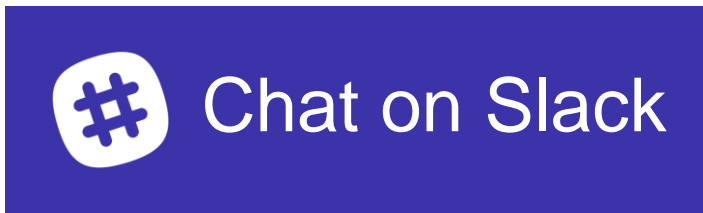
- Splunk is very popular for on premise deployments
- Splunk HEC provides a format that can be used with REST

Cloud Native

- Prometheus is popular for cloud native
- Lots of options here

Community Building

- Slack channel
 - Internal Slack channel allows for confidential exchanges of information
 - Public Slack for building a community
 - Lessons learned: Slack is not really designed for community building.



Example of how to register users who wish to join your Slack channel:

<http://ibm.biz/vgwslacksignup>

- StackOverflow tag
- GitHub repos with samples
- Dark launch of a public beta
 - Allowed both external and internal customers to access our Docker images

Conclusions

Think Microservices if you plan to shift to cloud native development

Plan to use modern dev ops tools to build it like Git, ZenHub, Jenkins, Gradle, etc.

Build your team to be as independent as possible (dev, support, etc.)

Utilize cloud container substrate for customer demos

Use container orchestration for HA and management of service

If going on premise, understand supported substrate options like CfC

InterConnect 2017



Notices and disclaimers

Copyright © 2017 by International Business Machines Corporation (IBM). No part of this document may be reproduced or transmitted in any form without written permission from IBM.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed “as is” without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.”

Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and

the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer’s responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer’s business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

Notices and disclaimers continued

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular, purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com, Aspera[®], Bluemix, Blueworks Live, CICS, Clearcase, Cognos[®], DOORS[®], Emptoris[®], Enterprise Document Management System[™], FASP[®], FileNet[®], Global Business Services[®], Global Technology Services[®], IBM ExperienceOne[™], IBM SmartCloud[®], IBM Social Business[®], Information on Demand, ILOG, Maximo[®], MQIntegrator[®], MQSeries[®], Netcool[®], OMEGAMON, OpenPower, PureAnalytics[™], PureApplication[®], pureCluster[™], PureCoverage[®], PureData[®], PureExperience[®], PureFlex[®], pureQuery[®], pureScale[®], PureSystems[®], QRadar[®], Rational[®], Rhapsody[®], Smarter Commerce[®], SoDA, SPSS, Sterling Commerce[®], StoredIQ, Tealeaf[®], Tivoli[®] Trusteer[®], Unica[®], urban{code}[®], Watson, WebSphere[®], Worklight[®], X-Force[®] and System z[®] Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.