76 Point Checklist
Making great DevOps teams
Our Story Starts with a Business that wants to grow and thrive

In 2013 started work on a new vision of the MSN Portal
MSN is Big

- #1 portal in 26 markets worldwide
- Half a billion monthly users worldwide
- 18B+ Monthly page views

$1 Billion a year business
Big Team

475 people in 4 locations Canada, India, Ireland, and United States
The Beginning of our Journey

Reaching our goal required a new technology stack
Old Gods - Old Ways Die Hard

Publishing System 10 years old

• Nightly reboots
• Fear of queue lengths
• Many single points of failure
New MSN, New Technology

Our business goals required a new tech stack

• Built for the cloud, using the latest technology
• Lower cost of ownership
• More reliable and scalable
Now it gets interesting

This is nothing like those other presentations on re-platforming
New Technology Stack

Lots of opportunity
Lots of risk

Eric Passmore http://www.ericpassmore.com Twitter:ericpassmore
Lots of Risk

• Entirely new “global” cloud with 4 datacenters
• New operating procedures for failure and recovery
Integration Risk

• System visibility required all new services built from the ground up
  • Co-developed new operational telemetry pipeline
  • Utilized ElasticSearch and Kibana for diagnostic logs
  • New dashboards, graphs, monitoring and alerting
Running At Scale Risks

- Scale required new pieces of infrastructure
  - Storage Service aggregating and sharding across 45 storage accounts
  - Document cache build from the ground up
  - New global topology with 4 datacenters and network of edge nodes
Risk From New Processes

- All new ways of doing things
  - New deployment processes
  - Coordination across new teams
  - Brand new set of tools
Then it gets crazier.
You can not make this up if you tried.
Launch All At Once

- Decision to flip all traffic over on a single day
- All regions, all markets, all users
- Billions of page views
- 200 Million users
Better Plan

• Decided to flight 10 percent of traffic for 1 month
• Other 90 percent still switched over in one day
I though I knew what to do.

I was very wrong
Standard Method

- Document First Level Dependencies
- Do Failure Mode Analysis
- Create a Health Model
Document Dependencies

**The Work**
- Inventory 1st level downstream services
- Ensure services have an SLA
- Create high level architecture

**Why It Failed**
- Too busy with tight timeline
- Seen as extra homework
- Teams did not know how the information would be used
- Lack of Trust
Failure Mode Analysis

The Work

- Brainstorm failures
- Score each failure
  - Score for impact the bigger the impact the larger the score
  - Score each failure for frequency the more frequent the larger the score
- Multiple impact and frequency
- Sort the list of failure by score.

Why It Failed

- Too tedious, torturous
- Tendency to focus on very rare and impactful events
- Exhaustive list of data dependent bugs
- Only finding the pain not solving the problem
Health Model

The Work
• Create a set of Mitigations
• Associate Mitigation with Failure Modes
• Create monitors and alerts to diagnose failure
• Automate mitigation as time and knowledge of system allows

Why It Failed
• Never got past failure modes
Birth of the Checklist

The only way out is up
Win a Fight without Fighting

• Needed a new approach
• Make DevOps Kung-Fu masters
  • Practical steps - no more theory
  • Race teams into shape
Birth of the Checklist

• Be Specific
• Four Categories
• Twenty two groupings
• 76 checklist items

Preflight checklist by US Air Force is licensed under a Creative Commons Attribution 4.0 International License
Four Categories

- Pre-Release: 13 of 76
- Monitoring: 29 of 76
- Deployment: 11 of 76
- Mitigation: 23 of 76
Pre-Release

Pre Check-in
1. Backward Compatible Schema and API
2. Versions for all assets in production

Load
3. Load Test – Memory leaks and race condition
4. Performance Tests – Latency under load
5. Exception Monitoring – keep a count
6. Capacity – enough capacity for peak traffic

World Ready
7. Globalization – one platform many markets

E2E
8. Testing user activity from start to finish

Eric Passmore http://www.ericpassmore.com Twitter:ericpassmore
9. Quality gates for partner services and content

10. Prevent URL manipulation
11. Prevent SQL injection
12. Prevent XSS

13. All pre-deployment processes are automated
Pre-Release: What Worked Best

• Building a single global platform unified support processes
  • Same tools used for all systems around the world
  • Same human process regardless of market
  • People developed trust with the system
  • Better team work from evidence based process

• Continuous capacity management
  • Built software to enable reporting across accounts & subscriptions
  • Enabled weekly capacity forecasts
  • Able to see degradations and react by adding hardware
Pre-Release: Improvement

• Not enough end to end scenario coverage
• Required manual tests and verification
• Half of all builds failed before production
• Bugs and Issues could have been found sooner
• Early discovery would be faster more powerful feedback loop

15. Staged Deployments – deploy a portion of the infrastructure and let it soak.

16. Deployment without service degradation

17. Patching – emergency hotfixes meet TTM goals

18. Monitor Deployments and auto-rollback to mitigate

19. Rollback to LKG – rollback take no longer than rollouts

20. Rollback to LKG – data rollouts require rollback
22. Post Deployment Smoke Tests – access to first level dependencies
23. Post Deployment Smoke Tests – Correctness of self, run http requests and test results
24. Post Deployment Smoke Test – Configuration correctness naming conventions match intended environment
Deployment: What Worked Best

• Confident change management from Automation & Practice of Releases
• Storing both v-previous and v-next of application for easy rollback
• Accepting principal of deployment without degradation
  • Did not get it right the first time, had some degradation
  • Key feedback loop enables rapid improvement
• Post-Deployment smoke tests
  • Pushed new code to small number of VMs
  • Ran tests before enabling live traffic
  • Found issues before production

Eric Passmore http://www.ericpassmore.com Twitter:ericpassmore
Deployment: Improvements

• Did not have enough diagnostics on deployments
• When deployments had trouble did not know why
• Needed to call in external support team
• Long cycle time to find and fix deployment issues
25. Actionable - monitoring must produce actionable, auto-escalated alerts

26. Tuning - If alert tuning can not be completed before service launch keep low level alerts and tighten over time

27. Alert on 5xx when rate exceeds threshold
28. Alert on 4xx when rate exceeds threshold

29. Alert on no-results (empty) when rate exceeds threshold

30. Alert on RPS when rate drops below or exceeds threshold

31. Alert when Queued requests are above threshold
32. Alert when application restarts exceed threshold
33. Browser Matrix - Monitoring includes the same suite of browsers as in pre-release

34. Global Coverage - Provide an alert for top markets when you create one for the US

35. Market Coverage - Provide market specific tests for market specific scenarios
36. Scenario Availability – use synthetic tests to ensure scenarios work

37. Broken Links – use crawler

38. Performance – monitor server latency or PLT for web pages

39. Last Mile – use Keynote to alert on last mile issues

40. Access – ensure endpoints are accessible

41. Raw counters – rely on raw counters to monitor downstream partners do not probe or use synthetic tests to measure availability of downstream partners
Telemetry Collection

System level time service aggregation
42. Processor usage
43. Memory usage
44. Network usage
45. Disk Usage (physical, logical)
46. Garbage Collection

Monitoring

Availablility Reporting

47. Services must automatically generate daily availability reports
48. Code Instrumentation – stack trace for errors

49. Standardize - Logging on inbound and outbound calls including time to receive response, headers sent and received, length of request and response

50. Correlation - Service should maintain and respect activity-id, the context attached to the request

51. Correlation - Service should propagate activity-id, the context to outbound requests

52. Correlation - Service should maintain logging for the duration of the request

53. Log Verbosity – Service should be capable of increasing and/or reducing logging verbosity on per-request basis
Monitoring: What Worked Best

- Drilling and Practice (Running races to get in shape)
  - We “pretended” we were live 2 months before launch
  - Live site issues had alerts and team responded
  - Team cohesiveness and decision quality dramatically increased
- One of the two most impactful things we did
Monitoring: Improvements

• Lots of probes for monitoring, instead of raw logs
• Probes (aka synthetic tests) were flaky
• Synthetic tests lacked diagnostic information
• Stand alone errors from synthetic tests were often ignored
• Clusters of errors were treated as real
• Lots of false alarms, lots of noise
54. Live Site Visualization Tools – Graphing time series data overlaying the same data from different sources

55. Live Site Diagnostic Tools – Specific tracing and debugging tooling

56. Stack Debugging – Tracking tools pinpoint a break across services

57. Basic Trouble Shooting Guides – written steps to diagnose and mitigate alerts
58. Advanced Trouble Shooting Guides – Explicit written procedures with clear thresholds for datacenter failover and designed to meet TTM targets

59. Readiness – complete online training for incident response

60. Cross-team escalations – ensure your team has contact information and they know how to use that contact information for functional escalations

61. Fire drills – Drill teams to practice procedures

62. Post Mortems – complete reviews and resulting work items for high impact incidents
63. Efficient Manual Failovers – Manual failover can be completed within 30 minutes

64. Automated Service Failovers – Traffic is automatically routed to healthy service endpoints to maintain service

65. Automated Partner Failovers – Comprehensive health monitor in place caches partner failure and triggers failover without human intervention.

66. Data Availability – All services must support active/active reads across datacenters

67. Sufficient Capacity to handle load due to failovers (redundancy)

68. Disaster Recovery Plan – Teams must complete annual verification that services meet objectives for availability and data recovery
69. Auto Retry on 4xx/5xx before aborting

70. Set SLA downstream – mutual agreement with partners and abort long running calls

71. Service Degradation – service degrade gracefully by suppressing specific experiences

72. Configure VIP health – automatically remove VM when service on that node is unhealthy
73. DOS prevention – set high water marks and throttle to prevent one misbehaving partner from impacting others

74. Auto-sizing – capacity is automatically added or reduced to handle peak traffic or failovers

75. Fault Injection – Double check monitoring, alerting, and mitigations during scheduled failure injection tests

76. Fault Injection – fail nodes under load to ensure service continues to run
Mitigation: What Worked Best

• Course grain fault injection
  • Closed gaps in monitoring and alerting
  • Took 6 DC failovers to get everything right
  • 6 practice runs created a well defined process

• Diagnostic tools
  • Tooling was the starting point for trouble shooting
  • Everyone worked off the same set of information
  • TraceId enabled cross stack investigation
  • Reproducible cases were easily handed off to downstream team
  • Better team work from being evidence based
Mitigation: Improvements

- Teams tended to diagnose root cause
- Teams did not follow pre-defined mitigations
- Time to mitigate from a business perspective was long
- Over time teams improved and accepted mitigations
Single Biggest Takeaway

• Do course grain failure injections
• Blow things up
• Large failures validate monitoring and alerting
• Large failures ensure system correctness in complex scenarios
• Large failures drill and prepare teams
Other teams took the checklist and created a Maturity Model

Wanted to convey progress not perfection

Eric Passmore http://www.ericpassmore.com Twitter:ericpassmore
Pre-Release

Code-Checks
- Peer Review of Code
- Unit tests coverage goals for Code
- Comprehensive Check-In Tests

Load
- Forked Production Traffic – shadow test

World Ready
- Content is localized – user preferences respected
- Content is localized – graceful fallback of preferences

UX
- Mobile layouts are correct for screen resolution – box model test
- Desktop layout is at parity – box model test
- Visual parity of key components – image compare
• Test flight once before scheduling
• Ramp up flights to validate impact
• Monitor flights for customer impact
• New flight numbers, do not reuse flight numbers
Monitoring

- Multiple Screens – Explicitly monitor mobile and table screens
Diagnostics

- Impact Analysis Tool – tools to calculate number of users or amount of revenue lost due to incident

Incident Management

- Post Mortem for medium severity incidents

Business Continuity

- All Data Services must support Active/Active writes
Feedback

• What did you like? What do you want to try?
• What would you do differently?