Game Day - Run Sheet

“Everyone has a plan until they get punched in the face”
- Mike Tyson

“You don’t choose the moment, the moment chooses you. You only choose how prepared you are when it does.”
Mike Burtch, Fire Chief SJFD

Whilst we have a run sheet, we must be prepared to go off script, and be comfortable with following these principles:

- Seek knowledge; the goal is learning, not following a script
- Ensure reproducibility; all tests start from a known, documented state (even if not from a “steady state”)

Time	What	Who	Duration
8:00 - 9:00	Setup room ● Fan, TV, whiteboard, internet, camera ● Print outs ○ Wireless access, test accounts etc. ○ GameDay reading materials and references to educate and inspire ○ Posters for passers by to understand what activity is taking place ● Prepare walls ○ Hypotheses facilitation ○ Retro wall ○ Feedback ○ Architecture diagram ● Setup conference bridge for remote attendees		60 mins

Introductions			
9:00	Introductions / Set the scene ● Goal is learning: ○ Find bugs, failures, design flaws ○ People and processes - can we improve diagnosis? ○ Build shared understanding ● Thanks to <externals/whomever> and those participating remotely ● Heavily rely on SMEs from each platform/team:		10 mins
9:10	Who's who in the zoo		10 mins
9:20	Logistics / Plan ● Room/Wall setup ● Run sheet and rough plan ● What to do when your component is not directly involved		10 mins
9:30am	Component role-call Checklist ❏ Environment up and running in XXX ❏ Environment connected to correct back-end ❏ Appropriate level of access (e.g. can make configuration changes as required) ❏ Visibility - can see logs, query DB, processes, dashboards appropriate for component ❏ Dashboards connected to TV/Monitors where required for the exercise Components ● Mobile ● API ● Database		15 mins

	• ...		
9:45	Confirm baseline end-user functionality is working (and therefore conducting a valid test)		15 mins
Scenario 1: XXXX			
10:00	Sub-scenario		30 mins
	Scenario description		
10:30	Sub-scenario		30 mins
Scenario 2: XXXXXX			
11:00			30 mins

11:30			30 mins
12:00			30 mins

Lunch			
12:30 - 13:30	EAT!		
Scenario 3:			
13:30			30 mins
14:00			30 mins

Scenario X: Deliberately reserved contingency / off-the-cuff tests			
15:00	Refer to curated contingency / stretch ideas on the wall		60 mins
Clean up activities			

16:00	Reverse any environment changes, e.g. ● Clean up any transactions ● Revert queue sizes ● Revert to production versions of environment ● ...		30 mins
Retro / Wrap up			
16:30 - 17:00	● Replay of findings ● Next steps on actions raised ● Review of retro wall ● Group photo		30 mins

Hypothesis Template \ Example

<table>
<tr><th colspan="3">Example hypothesis: isolated API failure</th></tr>
<tr><td>Scenario</td><td colspan="2">Failure in an API</td></tr>
<tr><td>Current State</td><td colspan="2">Running system with normal production load</td></tr>
<tr><td>Failure Recipe</td><td>Performed by:
DevOps</td><td>Kill a load balanced SE:

1. kill -9 <pid>
2. shutdown -r now
3. aws ec2 terminate-instances …</td></tr>
<tr><td>Hypothesis</td><td colspan="2">Description of expected events

1. Load balancer should mark nodes as down and stop sending traffic within x seconds / requests
2. 0 downtime
3. No alerts / monitoring should fire
4. In flight requests are safely aborted
a. Transactions in “pending” state (DB)

Questions to consider (modify to your specific needs, these are prompts)

<table>
<tr><th>Question</th><th>Answer</th></tr>
<tr><td>Customer impact?</td><td>A small percentage of customers (< x %) will experience a dropped transaction. Refresh and things should restore.

Killing Simple Payments whilst receiving a payment could result in an inconsistent state.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>Time to recover</td><td>< 30s</td></tr>
<tr><td>Expected Monitoring</td><td>1. Splunk should report API as down
2. Monitoring platform should send alerts to support team
3. Error rate should remain steady at X %
4. Email should be sent to DevOps \ Production Support</td></tr>
<tr><td>Alerts sent out?</td><td>Yes</td></tr>
<tr><td>Incident created?</td><td>Yes</td></tr>
<tr><td>Outage</td><td>No</td></tr>
<tr><td>DB state</td><td>Payments and transfers may result in an inconsistent state</td></tr>
<tr><td>Message Queue</td><td>Messages persisted and moved to dead letter queue</td></tr>
</table>
</td></tr>
<tr><td></td><td></td></tr>
<tr><td>Duration</td><td>30 minutes</td></tr>
</table>

Experiment process

1. Get into a known state
    a. for example, this could be a working system in a steady state with base ‘noise’ or a system in a known ‘failure’ state
2. Confirm hypothesis and failure we are going to introduce
    a. Flesh out details as required (document above)
3. Confirm who will check each component of the stack, and how to measure success/failure
4. Introduce failure
5. Monitor system
6. Capture any actions and document outcomes (can this test be automated?)
7. Explore any opportunities as they present themselves
8. Clean up