|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Health Monitoring** Ask yourself – how do you know if the software is even running? And if it is “healthy”? What about connectivity to databases, service busses or dependent web services? Is the performance within expected limits? What do you do with dead-lettered messages? Performance Monitoring How many? How often? How fast?  Analysing critical sections like external service calls and database queries is essential, and you can’t just rely on performance analysis dev tools.  You need **LIVE** performance data. |  |  | |  | | --- | | TL;DRDO  * Instrument your code * Log like your life depends on it * Provide monitoring metrics & health endpoints  DON’T  * Leave it until later  Contact T: @Rammesses E: [joel@hammond-turner.org.uk](mailto:joel@hammond-turner.org.uk) W: <http://nuggets.hammond-turner.org.uk> | | |  |  |  | | --- | --- | --- | |  |  | Imperium, Imperial Way,  Reading. RG2 0TD | | |  |  | |  | | --- | |  | |  | | Logging & Monitoring | | Making DevOps Happy | |
| **IT’S JUST PUSHING THE DEPLOY BUTTON, RIGHT?** It’s all about the “Ops”.  It’s **understanding** your software in a production environment.  Infrastructure affects your software, and equally your software affects the infrastructure on which it runs.  In a high-availability, load-balanced setup, how DO you detect a failure?  And in a virtualized environment, one service can affect others in the same VM estate. |  |  | **Exceptions** If you see exceptions in your logs, then you’ve found a scenario you’ve not coded for.  If you don’t capture as much exception data as possible, you’re never going to be able to analyse the scenario and fix the problem.  Inner, aggregate and reflection/invocation exceptions need explicit handling so you don’t lose the inner exception data.  DO NOT IGNORE EXCEPTIONS Forensic Logging Logging from your application is the thing that both makes your Ops team happy that the system is functioning, and gives them immediate data to mine or analyse when there are problems. But if your logging is sporadic, inconsistent or doesn’t provide value, then it’s worse than useless.  Consistent use of logging level is vital as logs may be filtered by level.  Log as if you don’t have anything else. |  |  | Ask yourself - can you diagnose a problem from logging alone?  One line of logging may be all you’ll get.  Critical is adding context (and where possible a correlation key) to every log line. You need to be logging the following   * **ATTEMPTS** *“Started processing {message}”* * **DECISIONS** *“Decided to do {action} to {entity} because {reasons}”* * **PREPARATION** *“About to do {action} to {entity}”* * **SUCCESS** *“Succeeded in doing {action} to {entity}”* * **TIMINGS** *“Took {elapsedMs}ms to do {action} to {entity}”* * **COMPLETION** *“Finished processing {message}”* |